

Testing Object-Oriented Systems: Lessons Learned

Robert V. Binder
RBSC Corporation www.rbsc.com

Overview

- Lessons Learned
 - Design, automation, and process
- The State of the Art
 - Design, automation, and process
- The State of the Practice
 - Three levels
 - Testing can achieve world class quality

Lessons Learned

- Class/Cluster test design
 - Super/subclass interaction must be tested: test at flattened scope.
 - Design subclass test suites to re-run on superclasses.
 - Design superclass test suites to re-run on subclasses.
 - Test polymorphic servers for LSP compliance.

Lessons Learned

- Class/Cluster test design
 - Exercise each binding of a polymorphic server message
 - Test all parameters for generics
 - Test interface data flow of non-modal classes

Lessons Learned

- Subsystem/system test design
 - Control easily obscured or accidental
 - Complex dependencies between concrete state and message sequence
 - Hierarchic control in state-based subclasses
 - Mosaic modularity at larger scope
 - Model behavior with state machines; achieve transition cover or better

Lessons Learned

- Subsystem/system test design
 - Objects don't compose (easily)
 - Producer's framework should not be in consumer's test scope
 - Minimum system/subsystem test includes
 - Testing exceptions
 - Testing class associations
 - Testing use cases (requires testable content)

Lessons Learned

- Test Automation
 - Encapsulation and mosaic modularity decrease controllability and observability
 - Design-by-contract/assertions is the only practical counter-measure for inherent non-determinism and loss of testability

Lessons Learned

- Test Automation
 - Avoid stubs: increase scope of the IUT or test in bottom-up order
 - Design test harness to exploit the structure and particulars of the system under test
 - Complete app = app components + test components under CM control

Lessons Learned

- Process
 - Inspect for omissions and inconsistencies, test for everything else
 - Design for testability
 - Implement hierarchic architecture patterns
 - Eliminate or encapsulate cyclic dependencies
 - Assert class invariants, at least
 - Support reuse with complementary producer/consumer testing strategies

Lessons Learned

- Process
 - 3 to 6 development increments
 - Developer class/cluster test -- Run tests locally, design tests globally: "unigration"
 - XP: "Continuous integration, relentless testing"
 - Independent build/integration group tests completed increment
 - Test suites must be regress-able
 - System testing on final increment

State of the Art

- Representation
- Design for Testability
- Test Design
- Test Automation

SOA: Representation

- Best Practices
 - Syntropy, Design by Contract
 - UML/OCL 1.0
 - Design Patterns
- Challenges
 - Architecture
 - Limits of cartoons
 - Test design as software engineering

SOA: Design for Testability

- Best Practices
 - Frameworks/libraries with assertions
 - Lakos' levelizable architecture
 - Percolation pattern
 - OS/400 test framework

SOA: Design for Testability

- Challenges
 - Seamless language support
 - OO testability an oxymoron?
 - Entropy horizon about 24 months

SOA: Test Design

- Best Practices
 - Test design patterns
- Challenges
 - Intra-class coverage
 - Polymorphic paths
 - Validated failure metrics/fault models

Test Design Pattern

- New pattern schema for test design

Name/Intent
Context
Fault Model
Test Model
Entry Criteria
Exit Criteria
Consequences
Known Uses

Representation
Test Generation
Oracle
Automation

*Testing Object-Oriented Systems: Models, Patterns,
and Tools. Addison-Wesley.*

Test Design Patterns

- Method Scope
 - Category-Partition
 - Combinational Function
 - Recursive Function
 - Polymorphic Message
- Class/Cluster Scope
 - Invariant Boundaries
 - Modal Class
 - Quasi-Modal Class
 - Polymorphic Server
 - Modal Hierarchy

Test Design Patterns

- Subsystem Scope
 - Class Associations
 - Round-Trip Scenarios
 - Mode Machine
 - Controlled Exceptions
- Reusable Components
 - Abstract Class
 - Generic Class
 - New Framework
 - Popular Framework

Test Design Patterns

- Intra-class Integration
 - Small Pop
 - Alpha-Omega Cycle
- Integration Strategy
 - Big Bang
 - Bottom up
 - Top Down
 - Collaborations
 - Backbone
 - Layers
 - Client/Server
 - Distributed Services
 - High Frequency

Test Design Patterns

- System Scope
 - Extended Use Cases
 - Covered in CRUD
 - Allocate by Profile
- Regression Testing
 - Retest All
 - Retest Risky Use Cases
 - Retest Profile
 - Retest Changed Code
 - Retest Within Firewall

SOA: Test Automation

- Best Practices
 - Design patterns for test automation
 - Automatic driver generation
 - Simple coverage analyzers

SOA: Test Harness Patterns

- | | |
|---|---|
| <ul style="list-style-type: none">• Test Case Implementation<ul style="list-style-type: none">– Test Case/Test Suite Method– Test Case /Test Suite Class– Catch All Exceptions• Test Control<ul style="list-style-type: none">– Server Stub– Server Proxy | <ul style="list-style-type: none">• Test Drivers<ul style="list-style-type: none">– TestDriver Super Class– Percolate the Object Under Test– Symmetric Driver– Subclass Driver– Private Access Driver– Test Control Interface– Drone– Built-in Test Driver |
|---|---|

SOA: Test Harness Patterns

- Test Execution
 - Command Line Test Bundle
 - Incremental Testing Framework (e.g. Junit)
 - Fresh Objects
- Built-in Test
 - Coherence idiom
 - Percolation
 - Built-in Test Driver

SOA: Test Automation

- Challenges
 - Validated failure metrics/fault models
 - Tool capability gaps
 - No support for OO-specific coverage
 - Very weak specification-based test generation
 - Weak support for test harness generation

State of the Practice

- Best Practices
 - Testing by scope (about 10%)
 - Many embedded/real-time shops
 - Extreme Programming
- Challenges
 - High-frequency/short cycle development
 - Naïve test design
 - Tool capability gaps

SOP: Testing by Poking Around

- About 70% of all organizations
- Characteristics
 - Testing done at developer discretion
 - No test entry/exit criteria
 - High tolerance for low quality

SOP: Testing by Poking Around

- Improvement Strategy
 - Assess limits of improvability
 - Train developers in basic test design
 - Install basic tool set:
 - Coverage analyzer
 - Memory leak detector
 - Test harness framework/generator (e.g. Junit)

SOP: Testing by Use Cases

- About 20% of all organizations
- Complies with "Unified Process" test approach
- Characteristics
 - Assumes objects "just work"
 - System test from use cases
 - Frustrated with chronic bugginess

SOP: Testing by Use Cases

- Improvement Strategy
 - Achieve exit criteria for indicated class/cluster test patterns
 - Use appropriate component/subsystem test design patterns.
 - Develop testable use cases
 - Implement test automation to support regression testing

SOP: Testing by Scope

- About one in ten
- Characteristics
 - Test design corresponds to scope
 - Scope-specific test entry/exit criteria
 - Effective test automation
 - Stable, repeatable process

SOP: Testing by Scope

- Improvement Strategy
 - Internal test design pattern-mining
 - Design for testability
 - Advanced test automation
 - Quantified closed loop feedback

Best Practice Examples

- Stepstone Corporation
- Ericsson CEE Project
- *Testing was the primary quality technique*

Stepstone Corporation

- ICpack 201 -- Objective-C class library
- Inspections for all classes
- Extensive automated test harness developed for each complex class
- No systematic test design

Ericsson CEE

- 75 KLOC C++ cellular support application
- Systematic testing at class, cluster, and system scope
- No other verification techniques used

Achieving World Class OO Quality

- Best-in-Class level:
 - An average of “less than 0.025 user-reported defects per function point” in the first year after release
- World Class = 10x Best in Class

Capers Jones. *Software Quality: Analysis and Guidelines for Success*.
(London: International Thompson Computer Press, 1997) p. 44

Achieving World Class OO Quality

| <i>Organization/ Language</i> | <i>KLOC</i> | <i>FP</i> | <i>Major Post Release Bugs</i> | <i>Bugs/FP</i> |
|-----------------------------------|-------------|-----------|--|----------------|
| Stepstone Objective-C | 12 | 414 | 5 | 0.0121 |
| Ericsson C++ | 75 | 1364 | 7 | 0.0051 |

Summary

- Lessons learned: OO testing requires unique test design approaches
- State of the art: expressed in patterns
- State of the practice: world class quality can be achieved

LESSONS LEARNED: TEST PROCESS IMPROVEMENT

*Martin Pol, IQUIP Informatica B.V.,
Wildenborch 3, 1110 AG Diemen, The Netherlands,
phone: +31 20 660 66 00, fax: +31 20 695 32 98
email: polmarti@iquip.nl, TPI[®]email: tpi@iquip.nl
TPI[®]-website: www.iquip.nl/tpi (in Dutch, English, German and Spanish)
TPI[®]-book authors: Tim Koomen and Martin Pol
(books available in Dutch, English and German (shortly))*

1. Introduction

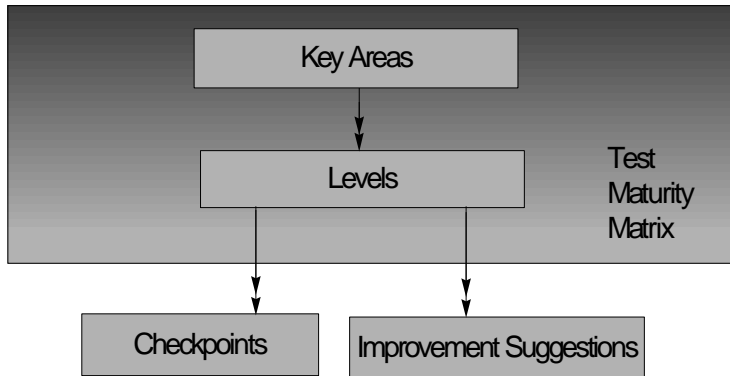
Apart from continuous evolutions in the technical infrastructure (e.g. Client/Server, GUI's and the Internet) and in the development infrastructure (e.g. OO, RAD) the challenges for testing lie in the integration of business processes and related innovations such as electronic commerce and computer telephony integration. We will connect customers, suppliers, employees, public services, etc., 24 hours per day, 7 days a week, world-wide. Complexity of the IT-solutions and testing will grow dramatically. The industry will focus on component integration, components of tailor-made software, packages, odd data files and incompatible technical infrastructure have to play together. Reusability and adaptability will be major quality requirements.

The growing need for testing and the ever changing challenges has made testing more and more mature. New testing methods, techniques and tools of a satisfactory level are developed, trained and experienced testing staff are more and more available and testing approaches which are generic to some degree enable organisations to deal with the continuous technology push. However, in practice it turns out to be hard to define what steps to take, and in what order, for improving and controlling the test process. A reference model for test process improvement is required to support the growing test maturity with the definition of small and controlled improvement steps.

Since Software Process Improvement models such as CMM/SEI generally are less specific for testing, suitable Test Process Improvement models are developed to enable structured test process improvement. The TPI[®] model, which is based on current state-of-the-art test process improvement practices, is an increasingly popular model that supports the selection and prioritisation of improvement steps. The purpose of such improvement could be reaching CMM level 3. The model also provides a large number of practical details and instructions on how to implement the improvement steps. Improving the test process is required to establish the stable testing foundation, which is a condition to face future challenges.

2. Description of the TPI[®] model

The model is visualised as follows:



2.1 Key Areas

In each test process certain areas need specific attention in order to achieve a well defined process. These **Key Areas** are therefore the basis for improving and structuring the test process. The TPI[®] model has 20 key areas.

The scope of test process improvement usually comprises black-box tests like system and acceptance tests. Most key areas are adjusted to this. However, to improve more "mature" test processes, attention must also be given to verification activities and white-box tests like unit and integration tests. Separate key areas are included in order to give due attention to these processes as well.

A full list of key areas is given below, followed by an explanation.

Test strategy
Life cycle model
Moment of involvement
Estimating & planning
Test design techniques
Static test techniques
Metrics

Test tools
Test environment
Office environment
Commitment and motivation
Testing functions and training
Scope of methodology
Communication

Reporting
Defect management
Testware management
Test process management
Evaluation
White-box testing

| Key Area | Description |
|------------------------|---|
| Test strategy | Test strategy should be aimed at finding the most important defects as early and as cheap as possible. In the test strategy it is determined what (quality)risks are covered by testing. By involving more tests and more detective measures, a better balanced strategy is possible. Unintentional overlaps or gaps between different tests can be prevented by co-ordinating testers and test activities, and by determining thoroughness. |
| Life cycle model | Within the test process a number of phases are discerned: planning, preparation, design, execution and completion. In each phase several activities are performed. For each activity aspects like goal, input, process, deliverables, dependencies, techniques and tools, facilities and documentation are recorded. The importance of a life cycle model lies in better control of the test process, since the activities can be planned and controlled consistently. |
| Moment of involvement | Although the actual test execution usually starts after the realisation of the software, the test process should start a lot earlier. Earlier involvement of testing in the system development life cycle helps detecting defects as early and/or as easy as possible, and even helps preventing defects. Better co-ordination between tests is possible and the critical path time of testing can be greatly reduced. |
| Estimating & planning | Estimating and planning are required in order to define which activities are performed at what moment and how many resources will be needed. Estimating and planning is the basis for reserving capacity and for co-ordinating test activities and project activities. |
| Test design techniques | A test design technique is defined as 'a standardised approach for deriving test cases from documentation'. Usage of these techniques increases insight in the quality and coverage of tests and leads to higher re-usability of tests. Based on a test strategy, different test design techniques are used in order to produce test coverage of the intended parts of the software to the extent which was agreed upon. |
| Static test techniques | Not everything can and needs to be tested dynamically, i.e. by running the programmes. The phenomenon of checking products without running the actual programmes or evaluating specified quality measures, is called static testing. Checklists and similar devices are very useful here. |
| Metrics | Metrics are quantified observations (measurements). For the test process, measuring the progress and the quality of the software under test is very important and so are metrics in these areas. Metrics are used in order to be able to manage the test process, in order to support advice to be given, and also in order to compare different systems or processes. It helps answering questions such as 'Why is it that system A has far less failures in production than system B has, why is it that test process A can be performed faster and more thoroughly than process B can?' In the improvement of the test process, metrics are specifically important for judging the results of certain improvement actions. This is done by measuring before and after the improvement takes place. |
| Test tools | Automation of the test process can be done in a variety of ways. As a rule, automation serves one of the following goals: - less resource consumption; - less time consumption; - better test coverage; - more flexibility; - more or faster insight in the status of the test process; - better motivation of test staff. |
| Testing environment | Test execution takes place in a so called test environment. This environment consists of the following components: <ul style="list-style-type: none"> • hardware; • software; • communication facilities; • facilities for creation and use of data sets; • procedures. The environment should be arranged so that optimal testing is possible. The environment greatly influences the quality, duration and costs of the test process. Important aspects of the environment are responsibilities, control, timely and sufficient availability, flexibility and representativeness of the actual production situation. |

| | |
|---|---|
| Office environment | Test personnel need offices, desks, chairs, PC's, word processing facilities, printers, telephones, etc. Good and timely arrangement of the office environment positively influences motivation of testers, and communication and efficiency of (the execution of) test tasks. |
| Commitment & motivation | Commitment and motivation of the people involved in testing are conditions for a mature test process. People involved are not only members of the test team, but also, amongst others, project managers and senior management. The test process is supplied with sufficient time, money and resources (both quantitatively and qualitatively) to perform a good test. Co-operation and communication with the others in the project results in an efficient process and in earlier involvement. |
| Testing functions & training | The test team requires a certain composition. A mixture is needed of different disciplines, functions, knowledge and skills. For example, apart from specific test expertise, also knowledge of the system under test is necessary, knowledge of the organisation and general knowledge of automation. Certain social skills are also very important. In order to get this mixture, education and training is needed. |
| Scope of methodology | For each test process a certain methodology or approach is used, consisting of activities, procedures, standards, techniques, etc. If these methodologies differ for each test process in the organisation, or if the methodology used is too generic, a lot of things have to be reinvented over and over again. The aim for an organisation is to use a methodology that is sufficiently generic to be widely applicable, but that has enough detail at the same time to be able to prevent undesired reinvention for each new test process. |
| Communication | In a test process communication takes place in a number of ways, both between testers as a group, and between testers and other members of the project, such as the developer, the end-user, the project manager. Topics of communication are test strategy, progress and quality of the software under test. |
| Reporting | Testing does not only deal with the detection of defects, but also with giving advice on (the lack of) quality of software. Reporting should be aimed at giving well funded advice to the project and customer on (the quality of) software and even on the software development process. |
| Defect management | Although defect management is in fact the project's responsibility, testers are strongly involved here. A good administration should be able to control the life cycle of a defect and to create several (statistical) reports. These reports are used to give well funded quality advice. |
| Testware management | Test products should be maintainable and reusable, and should therefore be managed. Apart from test products, also the products of prior phases, such as design and realisation, have to be managed well (although not by testers!). The test process can be seriously disrupted by delivery of wrong programme versions, etc. The demand of good management of these products, increases testability (and quality) of software. |
| Test process management | In order to manage each process and each activity, the four steps of the so called Deming circle are essential: plan, do, check, act. A well managed test process is of the utmost importance to perform the best possible test in the often very turbulent test arena. |
| Evaluation | Evaluation in this context means testing deliverables such as the functional design. In comparison to testing, the advantage of evaluation is the opportunity of early detecting defects. This causes the costs of repair to be considerably lower. Also, evaluation is relatively simple to establish since no programmes have to be run, no environment needs to be composed, etc. |
| White-box testing | A white-box test is defined as a test of the internal properties of the object, using knowledge of internal functioning. These tests are performed by developers. Well known white-box tests are the unit test and the integration test. Just like evaluation these tests take place earlier in the system development life cycle than black-box tests. Also, white-box tests are relatively cheap because less communication is required and because analysis is easier (the person detecting the defect is often the same person as the one who is to do the repairing. Besides, smaller objects are tested). |

2.2 Levels

The way key areas are organised within a test process determines the 'maturity' of the process. It is obvious that not each key area will be addressed equally thoroughly: each test process has its strengths and weaknesses.

In order to enable insight in the state of the key areas, the model supplies them with **Levels** (from A to B to C). On the average, there are three levels for each key area.

Each higher level (C being higher than B, B being higher than A) is better than its prior level in terms of time (faster), money (cheaper) and/or quality (better). By using levels we can unambiguously assess the current situation of the test process. It also increases the ability to advice targets for stepwise improvement.

Each level consists of certain requirements for the key area. The requirements (= checkpoints) of a certain level also comprise the requirements of lower levels: a test process at level B fulfils the requirements of both level A and B. If a test process does not satisfy the requirements for level A, it is considered to be at the lowest and, consequently, undefined level for that particular key area.

Below a description is given of the different levels of the key areas.

| Key Area | Levels | A | B | C | D |
|-----------------------------|---------------|--|---|--|--|
| Test strategy | | Test strategy for single test | Combined test strategy for black-box tests | Combined strategy for black-box tests plus white-box tests or evaluation | Combined strategy for all test and evaluation activities |
| Life cycle model | | Planning, Design, Execution | Planning, Preparation, Design, Execution, Completion | | |
| Moment of involvement | | Completion of specification | Start of specification | Start of requirements definition | Project initiation |
| Estimating and planning | | Fundamental estimating & planning | Statistically founded estimating & planning | | |
| Design techniques | | Informal techniques | Formal techniques | | |
| Static test techniques | | Intake test basis | Checklists | | |
| Metrics | | Project statistics (product) | Project statistics (process) | System statistics | Organisation statistics |
| Test tools | | Planning & control tools | Test execution & analysis tools | Integrated test automation | |
| Test environment | | Managed and controlled environment | Testing in most suitable environment | Environment on call | |
| Office environment | | Adequate & timely office environment | | | |
| Commitment and motivation | | Assignment of budget & time | Testing integrated in project organisation | Test engineering | |
| Test functions and training | | Test manager and testers | Support (methodical, technical, functional), control | Internal Quality Assurance | |
| Scope of methodology | | Project specific | Organisation, generic | Organisation, optimising (R&D) | |
| Communication | | Internal communication | Project communication (defects, change control) | Communication in organisation | |
| Reporting | | Defects | Progress (status of tests and products), activities (costs + time, milestones), defects with priorities | Risks & advice, including statistics | SPI advice |
| Defect management | | Internal defect management | Extended defect management, flexible reporting facilities | Project defect management | |
| Testware management | | Internal management & control of test deliverables | External management & control of test basis and test object | Reusable testware | Traceability: from requirements to test cases |
| Test process management | | Plan, do | Plan, do, check, react | Check, react in organisation | |
| Evaluation | | Evaluation techniques | Evaluation strategy | | |

| | | | | |
|-------------------|---|-----------------------------|-------------------------|--|
| White-box testing | Life-cycle: Planning, Design, Execution | White-box design techniques | White-box test strategy | |
|-------------------|---|-----------------------------|-------------------------|--|

2.3 Checkpoints

In order to determine levels, the TPI[®]-model is supported by an objective measurement instrument. The requirements for each level are defined in the form of **Checkpoints**: questions that need to be answered positively in order to classify for that level. Based on the checkpoints a test process can be assessed, and for each key area the proper level can be established. As each next level of a key area is considered an improvement, this means that the checkpoints are cumulative: in order to classify for level B the test process needs to answer positively to the checkpoints both of level B and of level A.

2.4 Test Maturity Matrix

After determining the levels for each key area, attention should be directed as to which improvement steps to take. This is because not all key areas and levels are equally important. For example, a good test strategy (level A of key area Test Strategy) is more important than a description of the test methodology used (level A of key area Scope of Methodology). In addition to these priorities there are dependencies between the levels of different key areas. Before statistics can be gathered for defects found (level A of key area Metrics), the test process has to classify for level B of key area Defect management. Such dependencies can be found between many levels and key areas.

Therefore, all levels and key areas are related to each other in a **Test Maturity Matrix**. This has been done as a good way to express the internal priorities and dependencies between levels and key areas. The vertical axis of the matrix indicates key areas, the horizontal axis shows scales of maturity. In the matrix each level is related to a certain scale of test maturity. This results in 13 scales of test maturity. The open cells between different levels have no meaning in themselves, but indicate that achieving a higher maturity for a key area is related to the maturity of other key areas. There is no gradation between levels: as long as a test process is not entirely classified at level B, it remains at level A.

| Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-----------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Key Area | | | | | | | | | | | | | | |
| Test strategy | | A | | | | | B | | | | C | | D | |
| Life cycle model | | A | | | B | | | | | | | | | |
| Moment of involvement | | | A | | | | B | | | | C | | D | |
| Estimating and planning | | | | A | | | | | | | B | | | |
| Test design techniques | | A | | B | | | | | | | | | | |
| Static test techniques | | | | | A | | B | | | | | | | |
| Metrics | | | | | | A | | | B | | | C | | D |
| Test tools | | | | | A | | | B | | | C | | | |
| Test environment | | | | A | | | | B | | | | | | C |
| Office environment | | | | A | | | | | | | | | | |
| Commitment and motivation | | A | | | | B | | | | | | C | | |
| Test functions and training | | | | A | | | B | | | C | | | | |
| Scope of methodology | | | | | A | | | | | | B | | | C |
| Communication | | | A | | B | | | | | | | C | | |
| Reporting | | A | | | B | | C | | | | | D | | |
| Defect management | | A | | | | B | | C | | | | | | |

| Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Key Area | | | | | | | | | | | | | | |
| Testware management | | | A | | | B | | | | C | | | | D |
| Test process management | | A | | B | | | | | | | | C | | |
| Evaluation | | | | | | | A | | | B | | | | |
| White-box testing | | | | | A | | B | | C | | | | | |

The main purpose of the matrix is to show the strong and weak sides of the current test process and to support prioritising actions for improvement. A filled in matrix offers all participants a clear view of the current situation of the test process. Furthermore, the matrix helps in defining and selecting proposals for improvement.

The matrix works from left to right, so low mature key areas are improved first. As a consequence of the dependencies between levels and key areas, practice has taught us that real 'outliers' (i.e., key areas with high scales of maturity, whereas surrounding key areas have medium or low scales) give little return on investment. For example, what is the use of a very advanced defect administration, if it is not used for analysis and reporting? Without violating the model, deviation is permitted, but sound reasons should exist for it.

In the example below, the test process does not classify for the lowest level of the key area test strategy (level < A), the organisation is working conform a life cycle model (level A) and the testers are involved at the moment when the specifications are completed (level A).

| Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-----------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Key Area | | | | | | | | | | | | | | |
| Test strategy | | A | | | | | B | | | | C | | D | |
| Life cycle model | | A | | | B | | | | | | | | | |
| Moment of involvement | | | A | | | | B | | | | C | | D | |
| etc. | | | | | | | | | | | | | | |

Based on this instance of the matrix, improvements can be discussed. In this example, a choice is made for a combined test strategy for black-box tests (=> level B) and for a full life cycle model (=> level B). Earlier involvement is at this moment not considered to be of relevance. The required situation is represented in the following matrix.

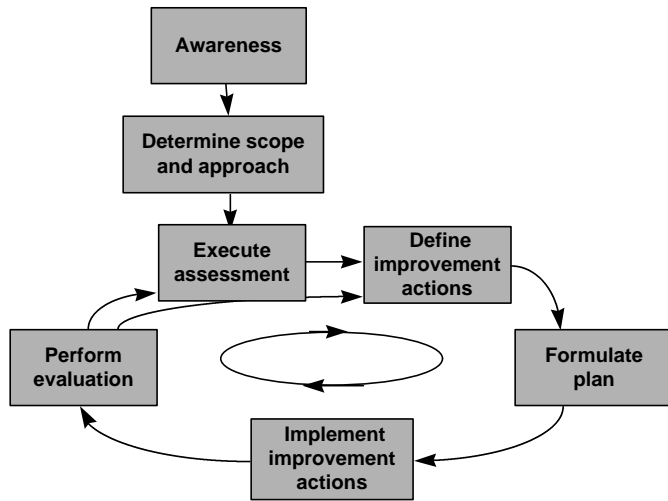
| Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-----------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Key Area | | | | | | | | | | | | | | |
| Test strategy | | A | | | | | B | | | | C | | D | |
| Life cycle model | | A | | | B | | | | | | | | | |
| Moment of involvement | | | A | | | | B | | | | C | | D | |
| etc. | | | | | | | | | | | | | | |

2.5 Improvement Suggestions

Improvement actions can be defined in terms of desired higher levels. For reaching a higher level the checkpoints render much assistance. Beside these, the model has other means of support for test process improvement: the **Improvement Suggestions**, which are different kinds of hints and ideas that help to achieve a certain level of test maturity. Unlike the use of checkpoints, the use of improvement suggestions is not obligatory. Each level is supplied with several improvement suggestions.

3. Application of the TPI[®] model

The process of test improvement is similar to any other improvement process. The figure below shows the various activities of an improvement process. These activities are discussed, with special attention for the places where the TPI[®] model can be used.



Awareness

The first activity of a test improvement process is to create awareness for the necessity to improve the process. Generally speaking, a number of problems concerning testing is the reason for improving the test process. There is a need to solve these problems and an improvement of the test process is regarded as the solution. This awareness also implies that the parties mutually agree on the outlines and give their commitment to the change process. Commitment should not only be acquired at the beginning of the change process, but be retained throughout the project. This requires a continuous effort.

Determine scope and approach

We determine what the improvement targets are and what the scope is. Should testing be faster, cheaper or better? Which test processes are subjects for improvement, how much time is available for the improvement and how much effort is it allowed to cost?

Execute assessment

In the assessment activity, an evaluation is given of the current situation. The use of the TPI[®] model is an important part of the assessment, because it offers a frame of reference to list the strong and weak points of the test process. Based on interviews and documentation, the levels per key area of the TPI[®] model are examined by using checkpoints, and it is determined which checkpoints were met, which were not met, or only partially. The Test Maturity Matrix is used here to give the complete status overview of the test process. This will show the strengths and weaknesses of the test process in the form of levels assigned key areas and their relative position in the matrix.

Define improvement actions

The improvement actions are determined based on the improvement targets and the result of the assessment. These actions are determined in such a way that gradual and step by step improvement is possible.

The TPI[®] model helps to set up these improvement actions. The levels of the key areas and the Test Maturity Matrix give several possibilities to define gradual improvement steps. Depending on the targets, the scope, the available time and the assessment results, it can be decided to carry out improvements for one or more key areas. For each selected key area it can

be decided to go to the next level or, in special cases, even to a higher level. Besides this, the TPI[®] model offers a large number of improvement suggestions which help to achieve higher levels.

Formulate plan

A detailed plan is drawn up to implement (a part of) the short term improvement actions. In this plan the aims are recorded and it is indicated which improvements have to be implemented at what time to realise these aims. The plan deals with activities concerning the content of the test process improvement as well as general activities needed to steer the change process in the right direction.

Implement improvement actions

The plan is executed. Because during this activity the consequences of the change process have the largest impact, much attention should be spent on communication. Opposition, which no doubt is present, must be brought to the surface and be discussed openly.

It has to be measured to what extent actions have been executed and have been successful. A means for this is the so-called "self assessment", in which the TPI[®] model is applied in order to quickly determine the progress. Here, the persons involved inspect their own test processes using the TPI[®] model.

Another vital part of this phase is consolidation. It should be prevented that the implemented improvement actions have a once-only character.

Perform evaluation

To what extent did the implemented actions yield the intended result? In this phase the aim is to see to what extent the actions were implemented successfully as well as to evaluate to what extent the initial targets were met. A decision about the continuation of the change process is made based on these observations.

4. Conclusions and remarks

Current developments proceed at a very high speed. The productivity of developers is rising continuously and the customers demand ever higher quality. Even if your current test process is fairly satisfactory, your process will need to improve in the future. The TPI[®]-model can help you with this.

The TPI[®]-model is an objective means to gain quick insight in the current situation of the test process. The model greatly offers help for improvement in the form of key areas, levels and improvement suggestions. It supports the definition of small and controlled improvement steps, based on priorities.

The reader might get the impression that use of the TPI[®]-model automatically leads to good analysis of the current and required situation. This is not true. The model should be seen as a tool for structuring the improvement of the test process and as a very good means of communication. Apart from the tool, improvement of test processes demands a high degree of knowledge and expertise of people involved, at least in the areas of testing, organisation and change management.

5. References

- Beizer, B. (1990), *Software Testing Techniques*, International Thomson Computer Press, ISBN 1-850-32880-3
- Bender, R. (1996), *SEI/CMM Proposed Software Evaluation and Test KPA*, STAR '96
- Boehm, B.W. (1979), *"Software Engineering Economics"*, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632
- Boeters, A., Noorman, B., (1997), *Kwaliteit op maat*, Kluwer Bedrijfsinformatie, ISBN 90-267-2579-5
- Burns, T., Stalker, G.M. (1995), *The Management of Innovation*, Oxford University Press, ISBN 0-19-828878-6
- Deming, W. Edwards (1992), *Out of the crisis*, University of Cambridge, ISBN 0-521-30553-5
- Emam, K. El (editor), Drouin, J. (editor), (1998), *Spice: The Theory and Practice of Software Process Improvement and Capability Determination*, IEEE Computer Society, ISBN 0-81867-798-8
- Ericson, T., Subotic, A., Ursing, S. (1996), *Towards a Test Improvement Model*, EuroSTAR '96
- Gelperin, D. (1996), *A Testability Maturity Model*, STAR '96
- Grady, Robert B., Caswell, Deborah L. (1987), *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, ISBN 0-13-821844-7
- Graham D., Herzlich, P., Morelli, C. (1996), *Computer Aided Software Testing, The CAST-report*, Cambridge Market Intelligence Limited, ISBN 1-897977-74-3
- Hall, Terence J. (1995), *The Quality Systems Manual : The Definitive Guide to the Iso 9000 Family and Tickit*, John Wiley & Sons, ISBN 0-471-95588-4
- Hetzel, W. (1993), *Making Software Measurement Work*, Wiley-QED, ISBN 0-471-56568-7
- Horch, John W. (1996), *Practical Guide to Software Quality Management*, Artech House Publishers, ISBN 0-89006-865-8
- Humphrey, Watts S. (1989), *Managing the Software Process*, Addison-Wesley, ISBN 0-201-18095-2
- Jarvis, A., Crandell, V. (1997), *Inroads to Software Quality*, Prentice Hall, ISBN 0-13-238403-5
- Juran, J.M. (1988), *Juran's Quality Control Handbook*, McGraw-Hill, ISBN 0-070-33176-6
- Kaner, C., Falk, J., Nguyen, H. Q. (1993), *Testing Computer Software (second edition)*, International Thomson Computer Press, ISBN 1-85032-847-1
- Kit, Edward (1995), *Software testing in the real world*, Addison-Wesley, ISBN 0-201-87756-2
- Kuvaja, P., Simila, J., Krzanik, L., Bicego, A., Saukkonen, S., Koch, G. (1994), *Software process assessment and improvement: the Bootstrap approach*, Blackwell

McFeeley, Bob (1996), *IDEALsm: a user's guide for Software Process Improvement*, Software Engineering Institute

Myers, G.J. (1979), *The Art of Software Testing*, Wiley-Interscience, New York NY10158, ISBN 0-471-04328-1

Perry, William E., Rice, Randall W. (1997), *Surviving the Challenges of Software Testing*, Dorset House Publishing, ISBN 0-932633-38-2

Pol, M., Teunissen, R., Veenendaal, E. van (1995), *Testen volgens TMap®*, Tutein Nolthenius, 's-Hertogenbosch, ISBN 90-72194-33-0

Pol, M., Teunissen, R., Veenendaal, E. van (1996), *Gestructureerd testen: een introductie tot TMap®*, Tutein Nolthenius, 's-Hertogenbosch, ISBN 90-72194-45-4

Pol, M., Veenendaal, E. van (1998), *Structured Testing of Information Systems: an Introduction to TMap®*, Kluwer, Deventer, ISBN.90-267-2910-3

Pulford, K., Kuntzmann-Combelles, A., Shirlaw S. (1995), *A quantitative approach to Software Management, the ami Handbook*, Addison-Wesley, ISBN 0-201-87746-5

Software Engineering Institute, Carnegie Mellon University (1995), *The Capability Maturity Model*, Addison-Wesley, ISBN 0-201-54664-7

Trienekens, J., Veenendaal, E. van (1997), *Software Quality from a Business Perspective*, Kluwer Bedrijfsinformatie, ISBN 90-267-2631-7

Books on TPI®:

Dutch:

Koomen, T. and M. Pol (1998), *Test Process Improvement®*, Leidraad voor stapsgewijs beter testen, published by Kluwer BedrijfsInformatie, The Hague

English:

Koomen, T. and M. Pol (1999), *Test Process Improvement®: a Practical Step-by-Step Guide to Structured Testing*, Published by Addison Wesley Longman, London

German:

Koomen, T. and M. Pol (1999), *Test Process Improvement®*, Anleitung für ein stufenweise optimiertes Testen, planned to be published shortly.

Internet:

at 'www.iquip.nl/tpi' several TPI®-products can be viewed and downloaded. Also questions can be asked and remarks can be made. Products are available in Dutch, English, German and Spanish.

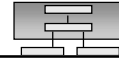


Lessons Learned: Test Process Improvement

Martin Pol



98 405 SCT 1



Agenda

- Lessons learned
- Future challenges
- Test Process Improvement



98 405 SCT 2





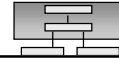
Why keep on testing ?

- Maturity of the industry
 - Error free software?
 - Prevention not enough
 - Technology push
- Business risks
 - Importance software quality
 - Integration business processes
 - Time-to-market
 - Competition

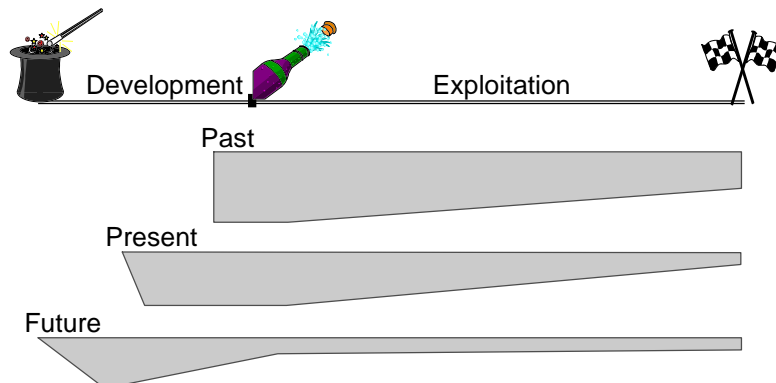
No Risk, No Test



98 405 SCT 3



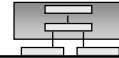
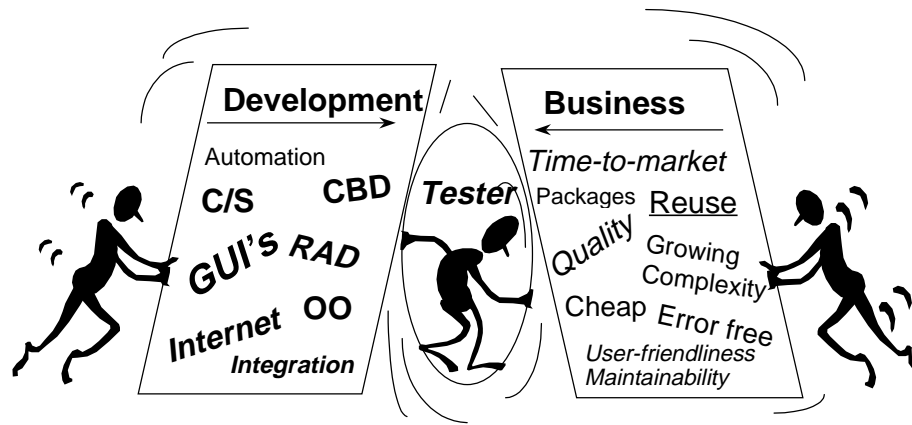
Testing: Past, Present and Future



98 405 SCT 4



Testing under pressure



Agenda

- Lessons learned
- Future challenges
- Test Process Improvement





Future Challenges

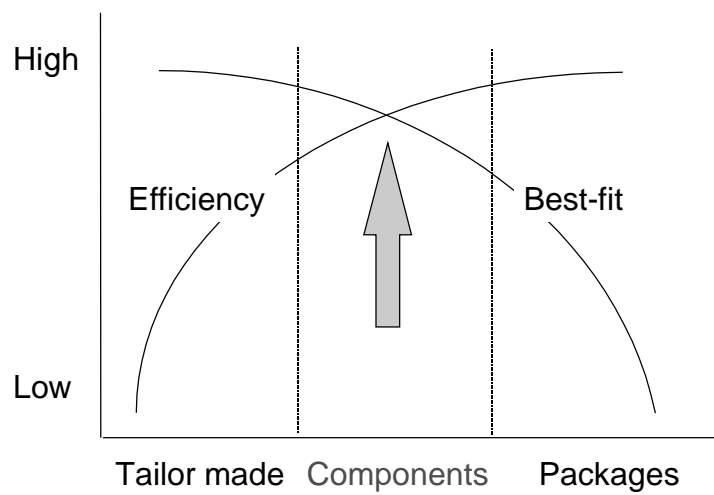
- Component Based Development
- Growing Complexity
- Suitable Test Methods and Tools
- Adequate Test Process Maturity



98 405 SCT 7



Component Based Development What's happening?



98 405 SCT 8





Growing complexity

back-office
front-office
customer
supplier
employee
government

hardware
software
networks
colour
graphics

e-commerce
virtual corporations
knowlegde mngnt
data mining
computer & telephony

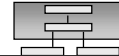
7 days a week
24 hours a day
world wide

art
sound
video
TV
smell?

object-orientation
client / server
GUI's
component assembly
Internet
.....



98 405 SCT 9



Suitable Test Methods and Tools

Available Use them!



98 405 SCT 10





Suitable Test Methods and Tools

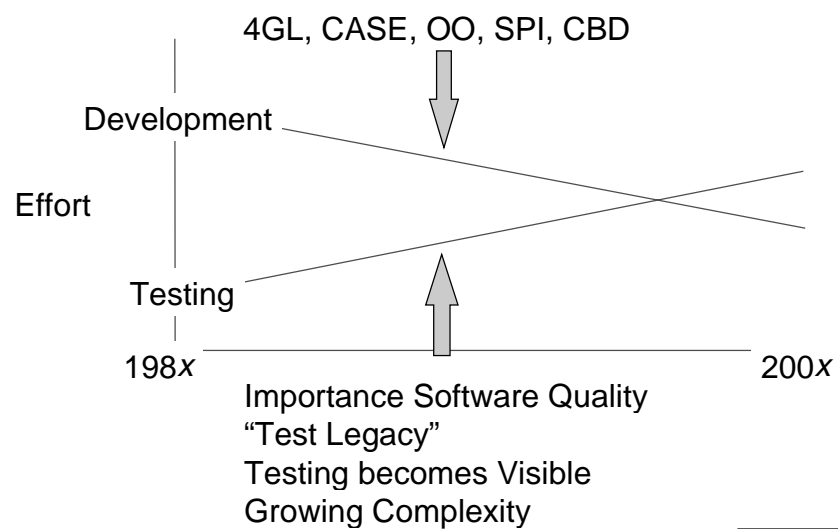
- Research & Development
 - Vendors & service suppliers
 - Universities, trendwatchers
 - In company
 - SIGIST's
- Training, conferences & publications
- Tester's professionalism
 - Skills and expertise for "whatever" test



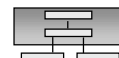
98 405 SCT 11



Unwanted situation

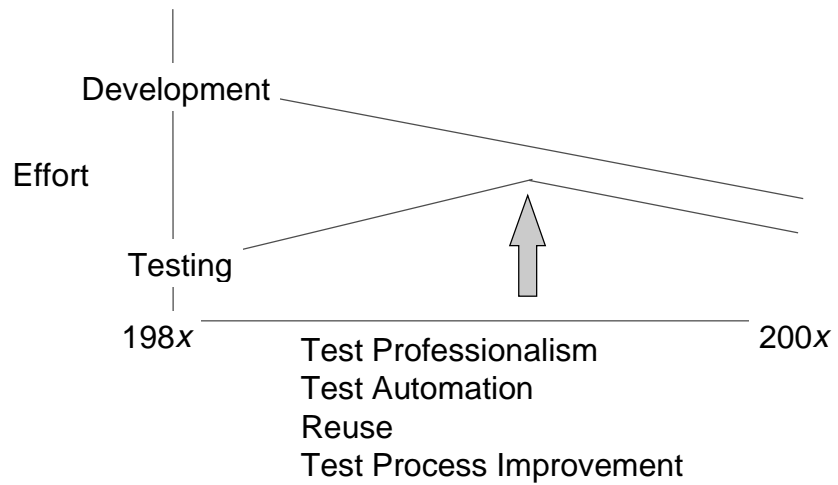


98 405 SCT 12

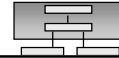




Desired situation



98 405 SCT 13



Agenda

- Lessons learned
- Future challenges
- Test Process Improvement



98 405 SCT 14





What is Test Process Improvement?

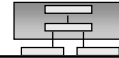
“Continuous improvement of the quality and the efficiency of the testing process, related to the output of the total software process”

- Quality
 - Insight
 - Coverage
 - Control
 - Timeliness
- Costs
 - Cheaper
- Lead time
 - Faster

**Required:
a reference model**



98 405 SCT 15



Model Requirements

- Controlled improvement steps
- Practical
- As objective as possible
- Options and priorities
- Highly detailed
- Fast assessment
- Independent

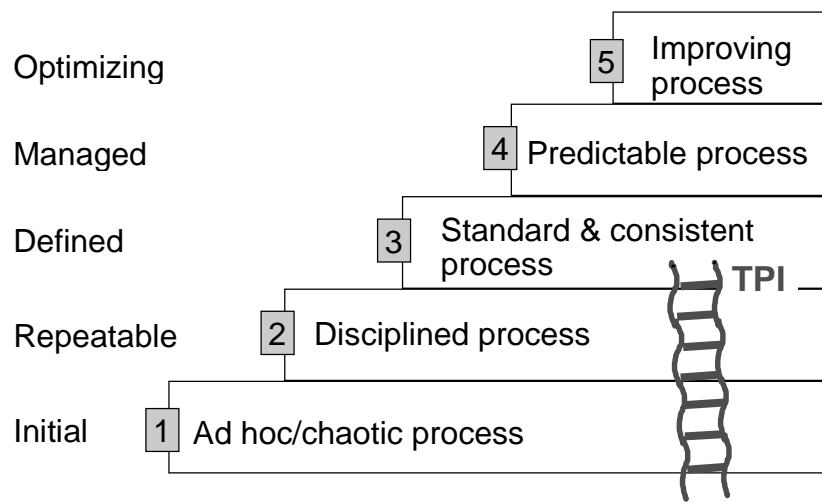


98 405 SCT 16





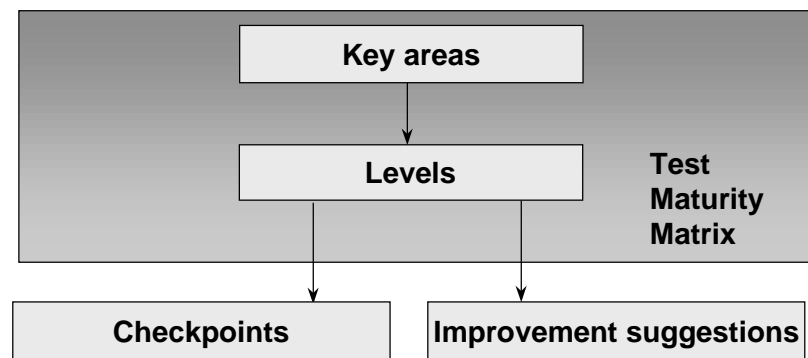
Capability Maturity Model and TPI



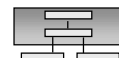
98 405 SCT 17



The TPI model



98 405 SCT 18



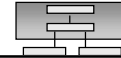


20 Key Areas

- Test strategy
- Life-cycle model
- Moment of involvement
- Estimating and planning
- Test specification techniques
- Static test techniques
- Metrics
- Test tools
- Test environment
- Office environment
- Commitment and motivation
- Testing functions and training
- Scope of methodology
- Communication
- Reporting
- Defect management
- Testware management
- Test process management
- Evaluation
- Low-level testing



98 405 SCT 19



Test Maturity Matrix

| Key area / Schale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 1 Test strategy | | A | | | | | B | | | | C | | D | |
| 2 Life-cycle model | | A | | | B | | | | | | | | | |
| 3 Moment of involvement | | | A | | | | B | | | | C | | D | |
| 4 Estimating and planning | | | | A | | | | | | | B | | | |
| 5 Test specification techniques | | A | | B | | | | | | | | | | |
| 6 Static test techniques | | | | | A | | B | | | | | | | |
| 7 Metrics | | | | | | A | | B | | | C | | D | |
| 8 Test tools | | | | | A | | | B | | | C | | | |
| 9 Test environment | | | | A | | | | B | | | | | C | |
| 10 Office environment | | | A | | | | | | | | | | | |
| 11 Commitment and motivation | | A | | | | B | | | | | | C | | |
| 12 Test functions and training | | | | A | | | B | | | C | | | | |
| 13 Scope of methodology | | | | | A | | | | | | B | | | C |
| 14 Communication | | | A | | B | | | | | | | C | | |
| 15 Reporting | | A | | | B | | C | | | | | D | | |
| 16 Defect management | | A | | | | B | | C | | | | | | |
| 17 Testware management | | | A | | | B | | | | C | | | D | |
| 18 Test process management | | A | | B | | | | | | | | C | | |
| 19 Evaluation | | | | | | | A | | | B | | | | |
| 20 Low-level testing | | | | | A | | B | | C | | | | | |



98 405 SCT 20



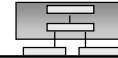


Current situation - example

| | Key area / schale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|----|-------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 1 | Test strategy | | A | | | | | B | | | | C | | D | |
| 2 | Life-cycle model | | A | | | B | | | | | | | | | |
| 3 | Moment of involvement | | | A | | | | B | | | | C | | D | |
| 4 | Estimating and planning | | | | A | | | | | | | B | | | |
| 5 | Test specification techniques | | A | | B | | | | | | | | | | |
| 6 | Static test techniques | | | | | A | | B | | | | | | | |
| 7 | Metrics | | | | | | A | | | B | | | C | | D |
| 8 | Test tools | | | | | A | | | B | | | C | | | |
| 9 | Testing environment | | | | A | | | | B | | | | | | C |
| 10 | Office environment | | | | A | | | | | | | | | | |
| 11 | Commitment and motivation | | A | | | | B | | | | | | C | | |
| 12 | Test functions and training | | | | A | | | B | | | C | | | | |
| 13 | Scope of methodology | | | | | A | | | | | | B | | | C |
| 14 | Communication | | | A | | B | | | | | | | C | | |
| 15 | Reporting | | A | | | B | | C | | | | | D | | |
| 16 | Defect management | | A | | | | B | | C | | | | | | |
| 17 | Testware management | | | A | | | B | | | | C | | | | D |
| 18 | Test process management | | A | | B | | | | | | | | C | | |
| 19 | Evaluation | | | | | | | A | | | B | | | | |
| 20 | Low-level testing | | | | | A | | B | | C | | | | | |



98 405 SCT 21



Desired situation - example

| | Key area / schale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|----|-------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 1 | Test strategy | | A | | | | | B | | | | C | | D | |
| 2 | Life-cycle model | | A | | | B | | | | | | | | | |
| 3 | Moment of involvement | | | A | | | | B | | | | C | | D | |
| 4 | Estimating and planning | | | | A | | | | | | | B | | | |
| 5 | Test specification techniques | | A | | B | | | | | | | | | | |
| 6 | Static test techniques | | | | | A | | B | | | | | | | |
| 7 | Metrics | | | | | | A | | | B | | | C | | D |
| 8 | Test tools | | | | | A | | | B | | | C | | | |
| 9 | Test environment | | | | A | | | | B | | | | | | C |
| 10 | Office environment | | | | A | | | | | | | | | | |
| 11 | Commitment and motivation | | A | | | | B | | | | | | C | | |
| 12 | Test functions and training | | | | A | | | B | | | C | | | | |
| 13 | Scope of methodology | | | | | A | | | | | | B | | | C |
| 14 | Communication | | | A | | B | | | | | | | C | | |
| 15 | Reporting | | A | | | B | | C | | | | | D | | |
| 16 | Defect management | | A | | | | B | | C | | | | | | |
| 17 | Testware management | | | A | | | B | | | | C | | | | D |
| 18 | Test process management | | A | | B | | | | | | | | C | | |
| 19 | Evaluation | | | | | | | A | | | B | | | | |
| 20 | Low-level testing | | | | | A | | B | | C | | | | | |



98 405 SCT 22





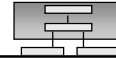
TPI Maturity Categories

| Key area / Schale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 1 Test strategy | | A | | | | | B | | | | C | | D | |
| 2 Life-cycle model | | A | | | B | | | | | | | | | |
| 3 Moment of involvement | | | A | | | | B | | | C | | D | | |
| 4 Estimating and planning | | | | A | | | | | | | B | | | |
| 5 Test specification techniques | | A | | B | | | | | | | | | | |
| 6 Static test techniques | | | | | A | | B | | | | | | | |
| 7 Metrics | | | | | | A | | B | | | C | | D | |
| 8 Test tools | | | | | A | | | B | | C | | | | |
| 9 Test environment | | | | A | | | | B | | | | | | C |
| 10 Office environment | | | | A | | | | | | | | | | |
| 11 Commitment and motivation | A | | | | | B | | | | | | C | | |
| 12 Test functions and training | | | | A | | | B | | | C | | | | |
| 13 Scope of methodology | | | | | A | | | | | | B | | | C |
| 14 Communication | | | A | | B | | | | | | | C | | |
| 15 Reporting | A | | | | B | | C | | | | | D | | |
| 16 Defect management | A | | | | | B | | C | | | | | | |
| 17 Testware management | | A | | | | B | | | | C | | | | D |
| 18 Test process management | A | | | B | | | | | | | | C | | |
| 19 Evaluation | | | | | | | A | | B | | | | | |
| 20 Low-level testing | | | | | A | | B | | C | | | | | |

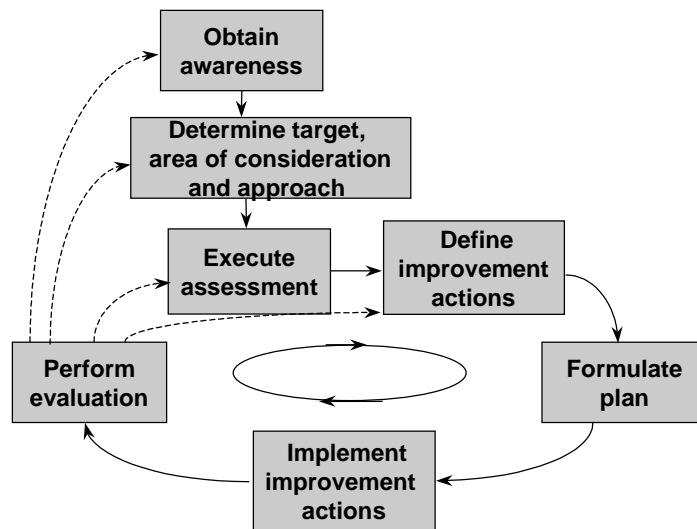
Project → Organization



98 405 SCT 23



Process of Change



98 405 SCT 24





Don'ts

- Exclusive top-down or bottom-up
- Confine to training
- Unbalanced improvement
- Unsuitable pilots
- High-level tests only
- Test tools are 'the' thing
- Underestimation of implementation
- Too many promises raise false expectations



98 405 SCT 25



Info about TPI:

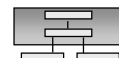
Books: English (available);
Dutch (available);
German (shortly)

Website: www.iquip.nl/tpi

Email: tpi@iquip.nl



98 405 SCT 26



Control Flow Animation

as a means of

Class Testing

by

Harry M. Sneed

Software Test Consultant

Arget, Bavaria

Fax: # 49-8104-669920

Email: Harry.Sneed@T-Online.de

for

Quality_Week_Europe

Brussels

1999

Abstract

In this paper a tool supported method for the dynamic analysis of object-oriented programs is described. The method involved uses animation to navigate through the source classes and to simulate the actual function execution. The goal is to test the control flow logic without compilation and without data. It amounts to an automated code inspection technique with data flow analysis.

Boris Beizer [1], Robert Binder [2], Gail Kaiser [3], Robson [4] and others [5] have pointed out the difficulties of testing object-oriented systems. Encapsulation means that object data is not accessible from outside the capsule. Distribution means that the data is distributed throughout the system and that it is allocated dynamically. Code inheritance makes it difficult to test derived classes without the superordinate base classes. Polymorphism prohibits being able to statically define paths through the system and forces the tester to consider all potential server functions. With the exception of encapsulation, these features discourage class testing. Probably the greatest obstacle to class testing is inheritance since it prohibits testing classes independently of one another. Another obstacle is the sheer number of foreign methods referred to. Many classes e.g. control classes, tend only to invoke methods in other classes. If these classes are not included in the test, their methods must be represented by stubs. All of these obstacles combine to discourage OO-developers from even trying to test their classes.

QW-1

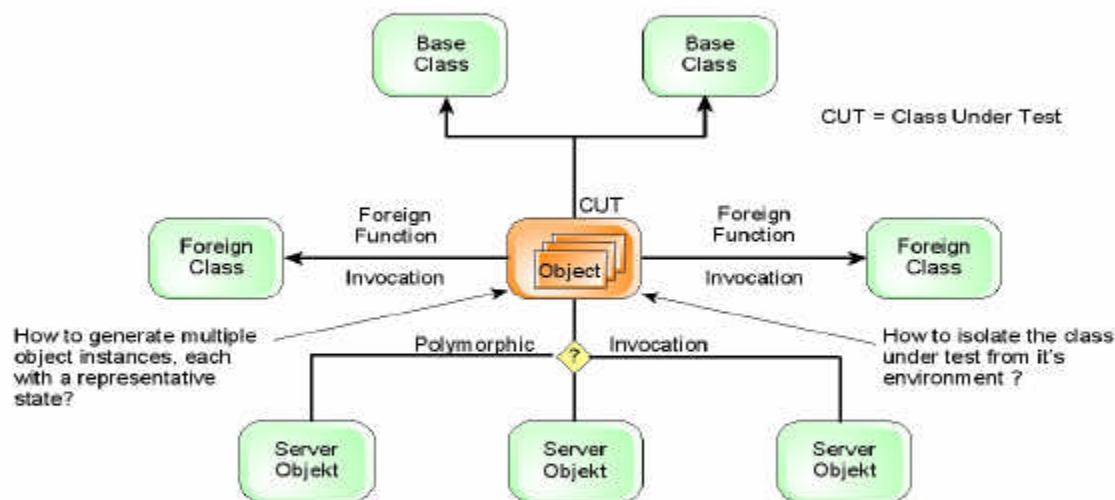


Fig. 1: Difficulties of Testing

The experience made by the author in a large scale object-oriented development project only confirms the observations made above. The whole system encompasses more than 3600

classes with 13 inheritance levels and 1.250.000 lines of code. Almost all of the subordinate classes are dependent on other classes at a higher level. Since multiple inheritance has been used frequently, the degree of dependence is even higher. The fact that any one class implementation may invoke up to 200 foreign methods makes independent class testing impossible. No more than five different class testing tools have been tried, including Cantata, Testbed and the McCabe tool, but none of them were accepted by the class developers. Either they are not able to cope with the complexity or they require too much effort to set up and use.

As a result of the difficulties involved in independent class testing, the author, who is responsible for quality assurance, decided to take another approach, an approach which would work without data at the source level. The new approach is based on source code animation. This paper describes that approach and how it contributes to error detection.

Source Code Animation

Animation is not new to software development. It has already been used in state charts to visualize the dynamics of parallel processing by stepping through the events. [6] It has also been used in CASE tools like SELECT to simulate data and process flow. [7] It has been particularly useful in simulating business processes. Time and again, animation has been used to step through the nodes of program control graphs. In this way, it gives the programmer the opportunity to validate the control logic. [8]

In current procedural code animators CALLs to other modules are generally ignored or simply recorded. This doesn't detract from the value of the animator since in procedural modules the number of calls to foreign modules is limited. However in object-oriented programs function calls make up a good part of the code. If they are not handled, the animation becomes useless. Therefore, the goal of animation in class testing must be to follow the control flow from one function to another across class boundaries. This will give the tester the possibility of tracing object interactions as depicted in the UML sequence diagram. [9]

Source code animation is implemented as an extension to static analysis. In static analysis all of the statements of a source code member, i.e. the class implementation, are identified and classified, including the function definitions. The tester must only identify where he wants to enter the class under test. This can be done by positioning the cursor on a particular line. The animator then steps through the code from statement to statement, displaying the current statement in a scroll bar until it comes to a decision or a function reference, at which point a pop up window is displayed.

When arriving at a decision node the user is requested to make a choice. In the case of an IF the answer is "true" or "false". In the case of a loop the answer is to continue or not. In the case of a switch statement, the user must select which CASE branch is to be taken. The stepping through of the statements is then continued from there.

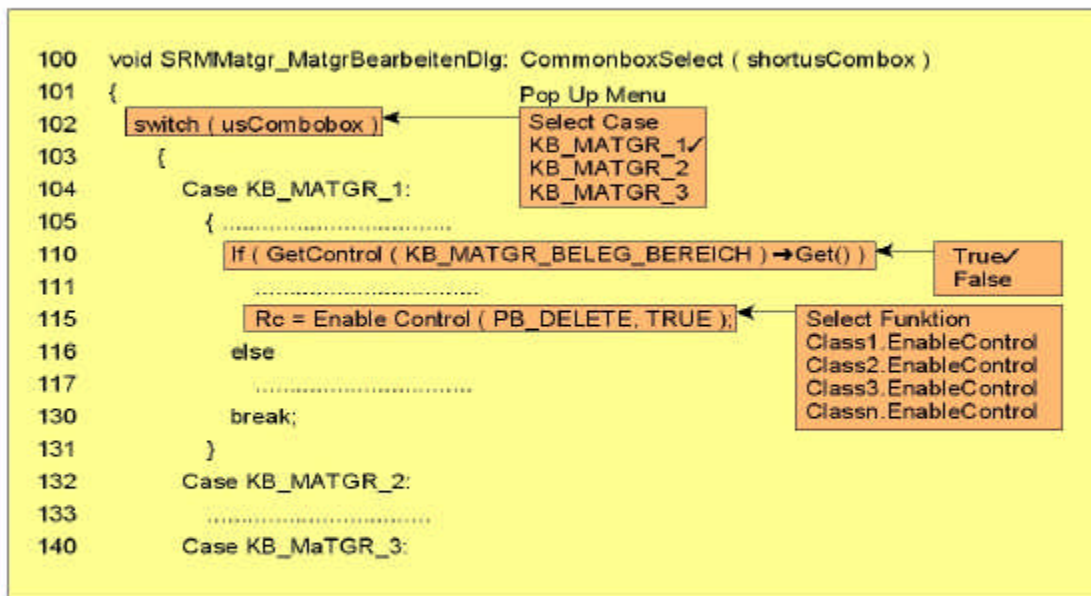


Fig. 2: Control Flow Step Thru

When arriving at a function reference, the user is asked whether the function should be included in the test or not. If not the animator skips over it to the next statement. If yes, the animator must look up in a table in which source member, i.e. class, the function is located. Then it loads that source member and continues the animation at the function indicated. A return statement causes the animator to go back to the next statement in the original code.

In this manner, a tester can step through nested layers of source code proceeding statement by statement, until she reaches the final return or until she terminates the animation. Since the user controls the step through process by mouse click or control key, she can also interrupt it at any time and also back up to a previous statement.

Animation Implementation

Source code animation is relatively easy to implement provided one already has a static analyzer. In this case, the author has already written a static analyzer called CPPANAL which was described in a previous paper. [10] The analyzer provides a statement table with references to the line number of the statement itself as well as line numbers of functions referenced. There is also an indicator of the statement type such as if, for, switch, while, etc.

First the tables are generated by a static analyzer and then turned over to a dynamic analyzer which interprets them. Using the statement table it is relatively easy to follow the control flow

and to move the scroll bar through the source text. Decisions and function calls are readily recognized. Besides the statement table for each source, there is a function table which indicates in which source member the functions are located. In the case of polymorphic functions, the function can be located in many different classes. Here the user must decide which one to branch to. The only problem that may arise is when a function referenced is in a foreign class that has not been analyzed yet. Then the user is given an error message and the animation continues in the original source.

The function table indicating where functions are located is a set of binary tuples

<function-name> : <source-name>

stored in an index table. The statement tables are hashed tables in core memory which are read in sequentially when referenced. Since source members in C++ seldom exceed 2000 lines, there has never been a size problem.

QW-3

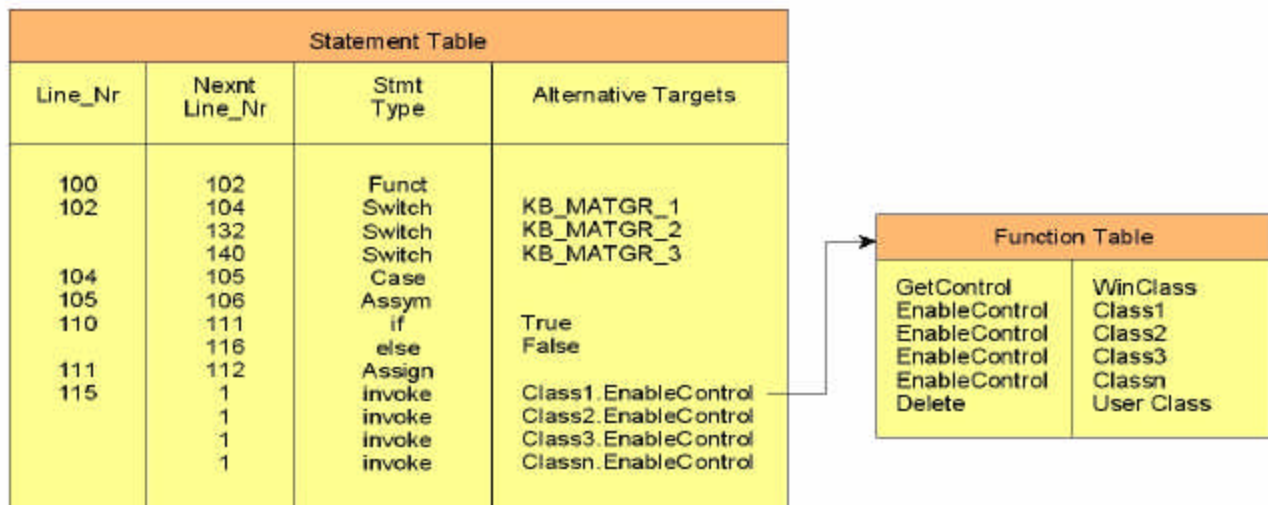


Fig. 3: Statement & Function Tables

Benefits of Source Code Animation

There are, of course, many who claim that since nothing is really executed, there is no real benefit except to familiarize the user with the source code. Besides the fact that this is, in itself, an important means of program comprehension there are other benefits as well.

For one thing, there is the opportunity of following the control flow through functions seeing the effects of conditional statements and loops and visualizing the sequence in which expressions are resolved. Here it may occur to the tester that certain conditions are wrongly formulated and that she may never get out of a loop.

Another benefit is to see what foreign functions are used and which ones are never reached. By marking the functions referenced, the animator is able to list out the functions not referenced at the end of the session. This is a means of identifying dead code.

Furthermore, it becomes visible to the tester at the source level how functions interact and in what order the interactions take place. As a consequence, the tester is able to detect sequence errors. The animator generates during the session a dynamic sequence diagram which can be compared to the original static one. [11]

QW-4



Fig. 4: Dynamic Sequence Diagram

Finally, since the statement table also contains the expressions with their results, it is possible to produce a data flow table containing the operations performed upon given results. This comes close to symbolic execution as defined by Howden back in the 1970's. [12] Each significant output variable is assumed a path condition which defines the conditions and expressions to be executed to attain a given state.

```
Interest =: Account.Balance*Current->Interest-Rate
           if(Account.Balance>0) && (Validate.Account)
           if(Current->Date==Account.Interest.Date)
           if(This.Account==Sarmysaccount)
```

QW-5

```
Rc = EnableControl ( PB_DELETE, TRUE )
     If ( GetControl ( KB_MATGR_BELEG_BEREICH ) → Get ( ) )
     Case usCombobox = KB_MATGR_1
     In Function: ComboboxSelect (short usCombobox)
     In class: SRMMatgr_MatgrBearbeitenDlg
```

Fig. 5: Path Expression for RC

Justification of Animation

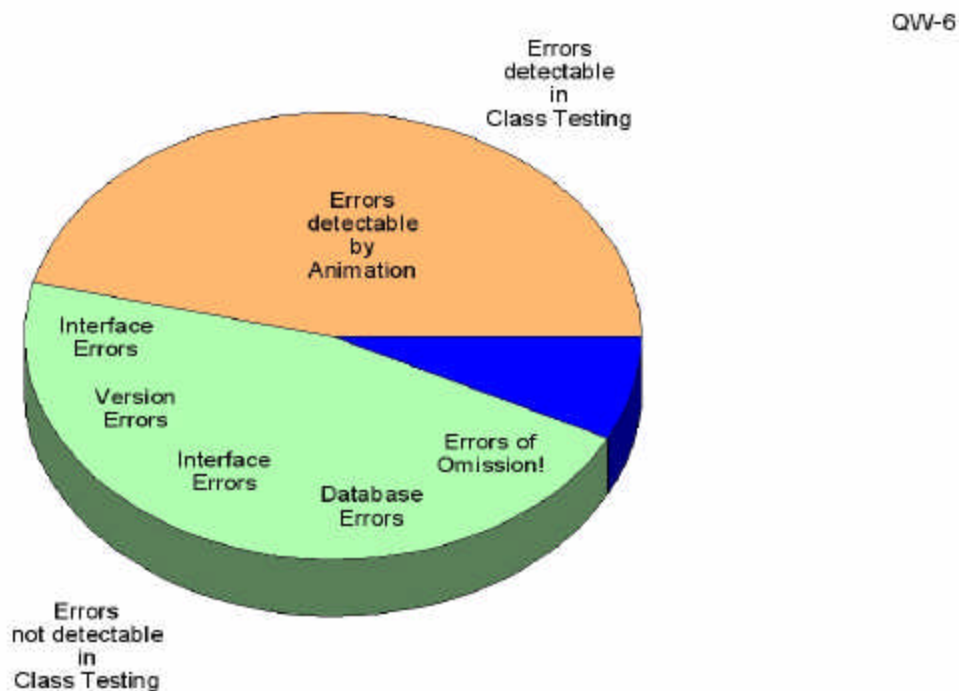
When it comes to testing classes which are dependent on many other classes in a complex class hierarchy, there may be no better method of **detecting** errors in the source than through animation. The problem with sophisticated class testbeds is that they often hide problems and are of themselves a source of error if not used properly. In stubbing out function calls the user is forced to simulate the return results of the functions called which may be complex objects or pointers to complex objects. Then there is the whole problem of inheritance which makes it so difficult to test derived classes without the base classes. If used extensively, inheritance prohibits single class testing.

In light of the difficulties involved in setting up an adequate class test environment animation may prove to be an economic compromise which is easy to implement, easy to use and still useful in detecting certain type of errors, i.e. the same types of errors that are found when actually executing the modules.

Conclusion

In 1978, this author and Dr. Ed Miller of SRA, San Francisco set up the first European testing factory in Budapest, Hungary. The purpose of this factory was to test SPL modules for a large scale Siemens project on a fixed price basis. As reported on at the Florida Workshop on Software Testing in Dec., 1978 only 48 % of the total errors found were actually discovered in module testing. [13] The rest were either interface errors, environment errors or errors of omission. In module testing, mainly logic, initialization and computation errors were detected. In almost all cases these errors were discovered while analyzing the code to specify test cases prior to actually testing the module, i.e. the test execution was only a confirmation of the assumption that an error existed.

With the aide of source code animation the same results can be achieved with much less effort. Thus, for the 48 % of errors that may be detected in module testing, animation would suffice to uncover at least 80 %. [14] That is more than enough to justify using this approach.



Finally, there is something often overlooked by test tool developers in their obsession to find a perfect solution and that is, that their ingenious devices must be used by quite ordinary programmers working under extreme time pressure, who have little time for tinkering with sophisticated tools resembling a Rube Goldberg machine. Their job is to find as many errors as possible in the limited time available. Animation is a good way of achieving this goal.

References

- [1] Beizer, B.: "Testing Technology - The growing Gap" in American Programmer, Vol. 7, No. 4, April 1994, p. 3-11
- [2] Binder, R.: "Design for Testability in object-oriented Systems" in Comm. of ACM, Vol. 37, No. 9, Sept. 1994, p. 28-52
- [3] Perry, D./Kaiser, G.: "Adequate Testing and object-oriented Programming" in IOOP, Jan. 1990, p. 13-19
- [4] Smith, M./Robson, D.: "Object-oriented Programming - The Problems of Validation" in Proc. of int. Conf. on Software Maintenance - 1990, IEEE Press, San Diego, Nov. 1990, p. 272-281
- [5] Sneed, H.: "Objektorientiertes Testen" in Informatik Spektrum, No. 18/1, Feb. 1995, p. 6-12
- [6] Harel, D.: "Executable Object Modeling with Statecharts" in IEEE Computer, July 1997, p. 31-42
- [7] Allen, P./Frost, S.: "Component-Based Development for Enterprise Systems - The Select Perspective", Cambridge Press, Cambridge, 1998, p. 151-160
- [8] Stevens, S.: "Intelligent Interactive Video Simulation of a Code Inspection", in Comm. of ACM, Vol. 32, No. 7, July 1989, p. 832
- [9] Poston, R.: "Automated Testing from Object Models" in Comm. of ACM, Vol. 37, No. 9, Sept. 1994, p. 48-58
- [10] Sneed, H.: "Comprehending a complex, distributed, object-oriented Software-System" in Proc. of 7th Int. Workshop on Program Comprehension, Pittsburgh, PA, May 1999
- [11] Sidhu, D./Leung, T.-K.: "Formal Methods for Protocol Testing", in IEEE Trans, on S.E., Vol. 15, No. 4, April 1989, p. 413-426
- [12] Howden, W.: "Symbolic Testing with the DESSECT Symbolic Evaluation System" in IEEE Trans. on S.E. Vol. 1, No. 4, July 1977, p. 266-278
- [13] Budd, T./Majoros, M.: "Experiences in a Software Test Factory", in Proc. of IEEE Workshop on Software Testing, Dec. 1978, p. 112-137
- [14] Daran, M./Pacale, T.: "Software Error Analysis - A real case study" in Proc. of ISSTA-96, ACM Press, Jan. 1996, p. 158-171

Software Quality Week Europe
Brussel, November 1999

Control Flow Animation as a means of Class Testing

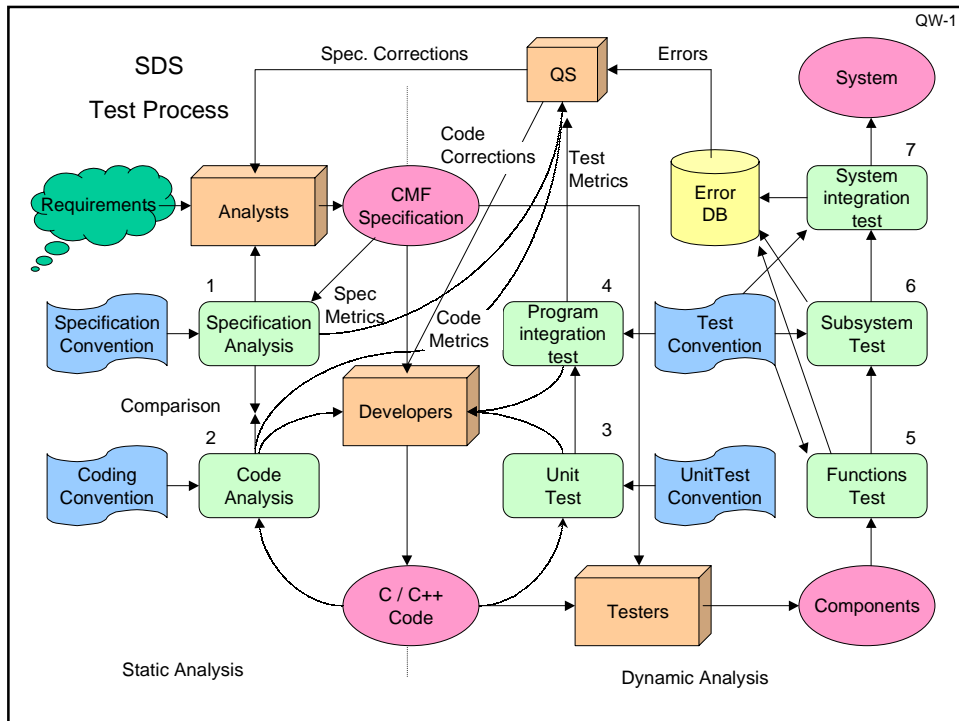
by

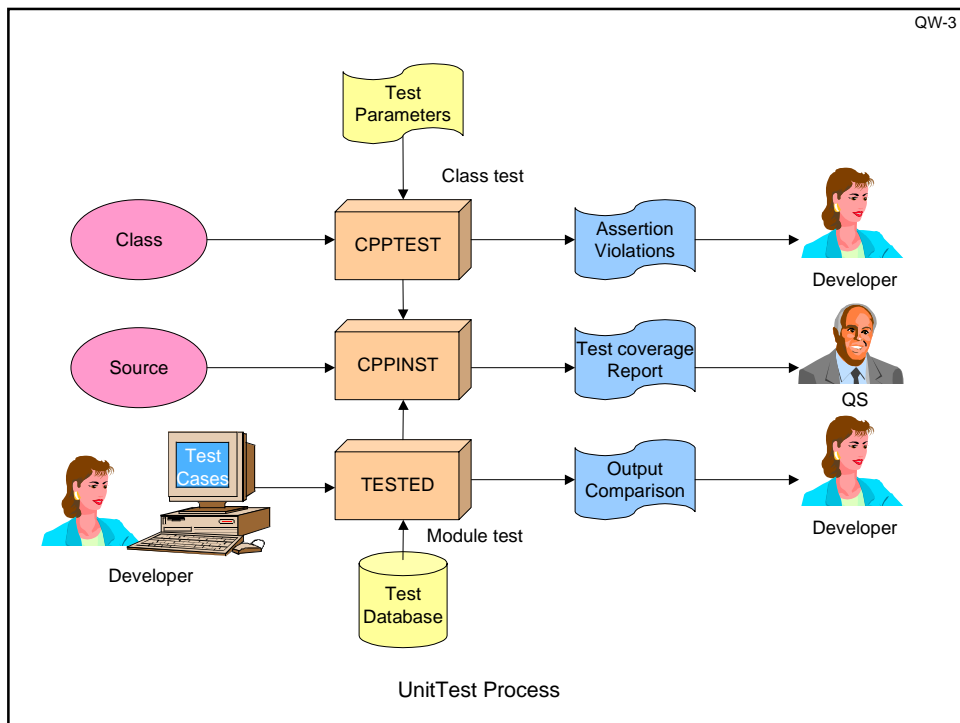
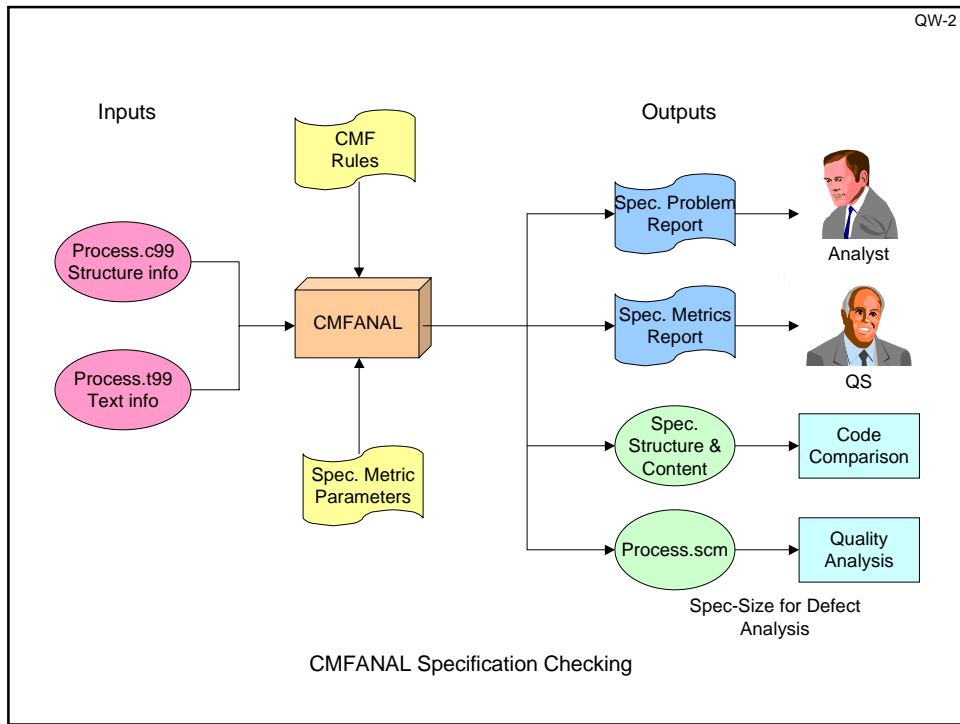
Harry M. Sneed

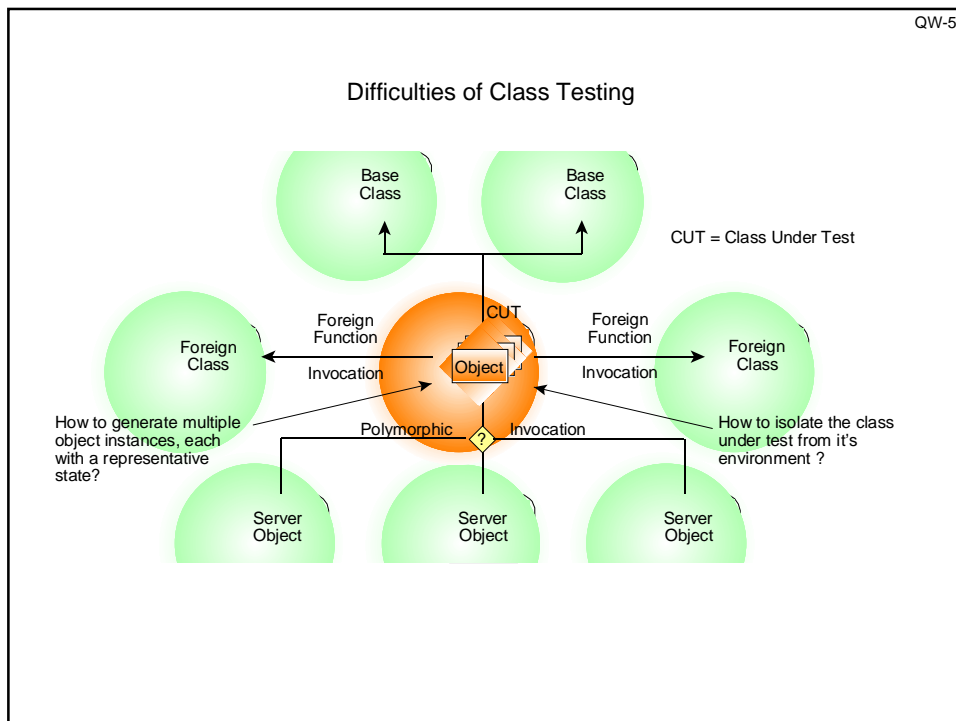
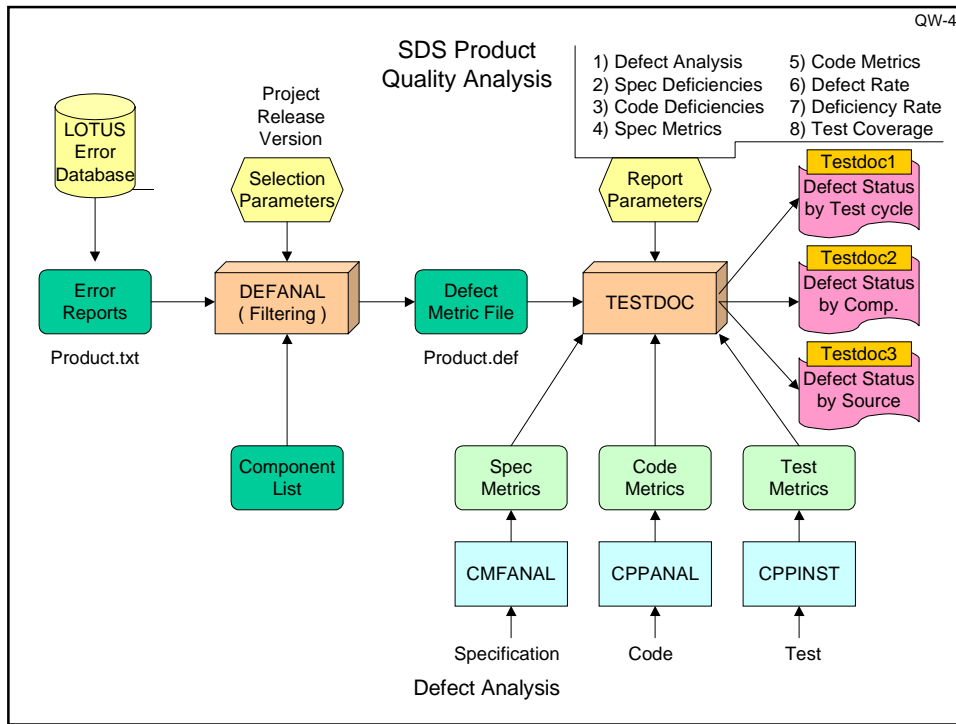
Software-Test Consultant
Fax #49-8104-669920
E-Mail:Harry.Sneed@t-online.de

Experience with the use of a source animation tool
in validating C++ classes

Topics:
The test process
Specification validation
Code checking
Class testing
Results obtained
Conclusions







Control Flow Step Thru

```

100 void SRMMatgr_MatgrBearbeitenDlg: CommonboxSelect ( shortusCombox )
101 {
102     switch ( usCombobox )
103     {
104         Case KB_MATGR_1:
105             {
110                 If ( GetControl ( KB_MATGR_BELEG_BEREICH ) @Get() )
111                 {
115                     Rc = Enable Control ( PB_DELETE, TRUE );
116                 }
117                 else
118                 {
130                     break;
131                 }
132             }
133         Case KB_MATGR_2:
134             {
140                 Case KB_MaTGR_3:

```

Pop Up Menu
 Select Case
 KB_MATGR_1*
 KB_MATGR_2
 KB_MATGR_3

True*
 False

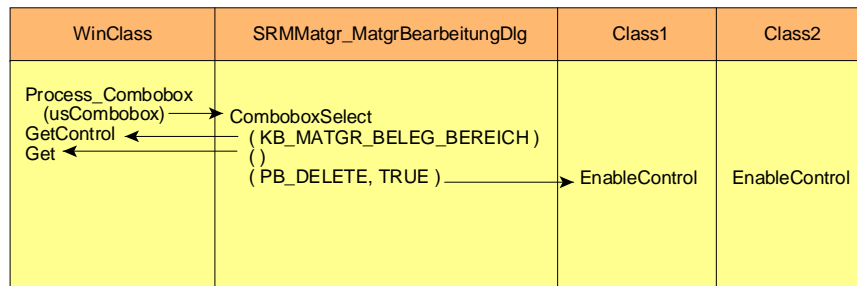
Select Funktion
 Class1.EnableControl
 Class2.EnableControl
 Class3.EnableControl
 Classn.EnableControl

Statement and Function Tables

| Statement Table | | | |
|-----------------|---------------|------------|----------------------|
| Line_Nr | Nexnt Line_Nr | Stmnt Type | Alternative Targets |
| 100 | 102 | Funct | |
| 102 | 104 | Switch | KB_MATGR_1 |
| | 132 | Switch | KB_MATGR_2 |
| | 140 | Switch | KB_MATGR_3 |
| 104 | 105 | Case | |
| 105 | 106 | Assym | |
| 110 | 111 | if | True |
| | 116 | else | False |
| 111 | 112 | Assign | |
| 115 | 1 | invoke | Class1.EnableControl |
| | 1 | invoke | Class2.EnableControl |
| | 1 | invoke | Class3.EnableControl |
| | 1 | invoke | Classn.EnableControl |

| Function Table | |
|----------------|------------|
| GetControl | WinClass |
| EnableControl | Class1 |
| EnableControl | Class2 |
| EnableControl | Class3 |
| EnableControl | Classn |
| Delete | User Class |

Dynamic Sequence Diagramm

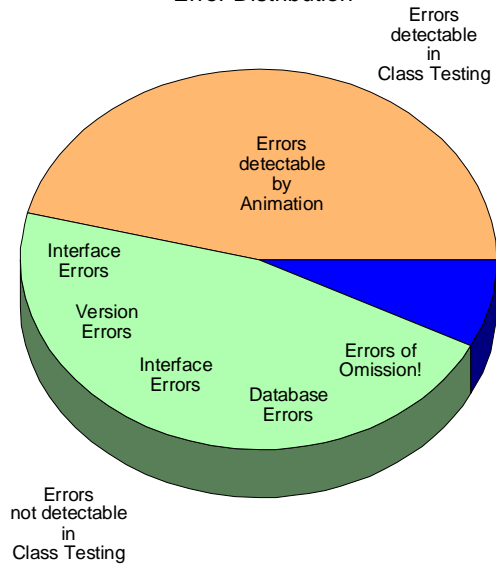


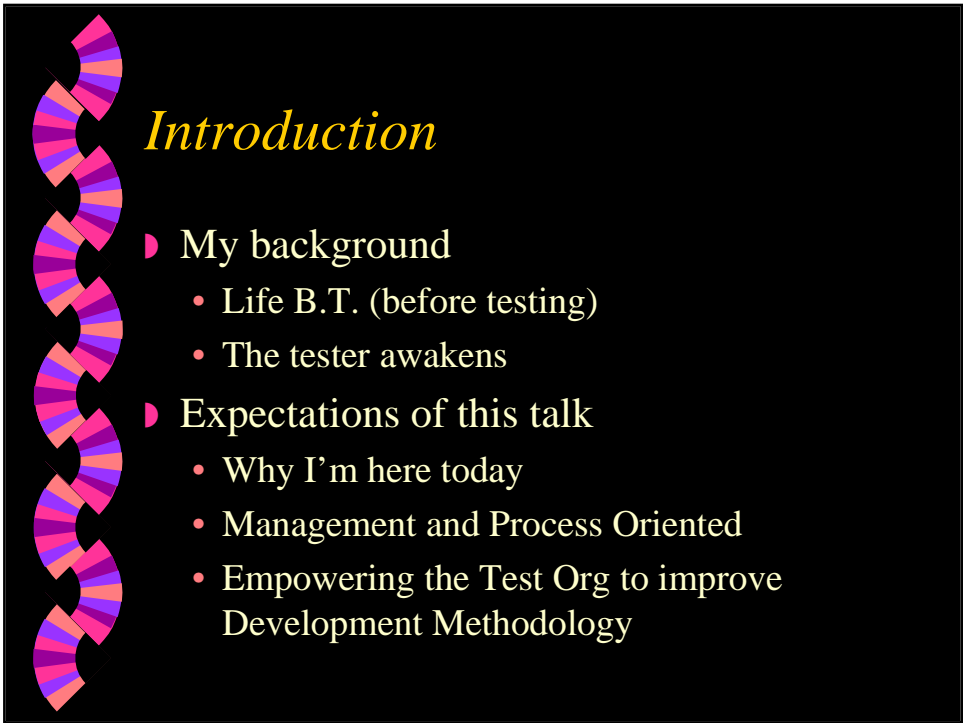
Path Expression for RC

```

Rc = EnableControl ( PB_DELETE, TRUE )
  If ( GetControl ( KB_MATGR_BELEG_BEREICH ) & Get ( ) )
    Case usCombobox = KB_MATGR_1
      In Function: ComboboxSelect (short usCombobox)
      In class: SRMMatgr_MatgrBearbeitenDlg
  
```

Error Distribution







Addressing our differences: Products & Paradigms

- ▶ The labels we live by
 - CMM, ISO, SEI, etc.
 - MIL-Spec
 - Shrink Wrap
- ▶ Common ground
 - We all want a better process
 - Higher quality
 - Accurate Schedules



Why should Testing drive this?

- ▶ Biggest stakeholder?
- ▶ Desire
- ▶ Pragmatism
- ▶ Can any one else do it better?
- ▶ Don't forget the Tester's Mission



Seizing Control: Step 1, Collecting Data

- In-depth Post-Mortem: Test
 - Various methods to collect data
 - Spend as much energy on your group as critiquing others
 - Important to reach consensus on items in conflict
 - Roll-up big ticket items from Cross-Functions only



Seizing Control Step 1: Collecting Data

- In-depth Post-Mortem, Cross-Functional
 - Powerful Process to collect both Best/Worst Practices
 - Techniques
 - Moderated Post-Mortems
 - Self-Administrated Post-Mortems
 - Frequency & Follow-up



Seizing Control Part 2: Analysis

- How can the current Development model hurt test?
 - Identify areas both from Post-Mortem and ones missed
 - How could these change?
 - How would change affect other teams?
 - Would these improve the chances for higher quality and earlier ship date?
 - Refine your "laundry list"



Seizing Control Part 3: Communicating Results

- Rollout of Changes to your Test/Quality Organization
 - Hash out with your leads/managers
 - Discuss changes with entire team
 - Pitfalls
 - Solving the right problem the wrong way
 - Process change overload
 - Cost/Benefit analysis



Seizing Control Part 3: Communicating Results

- Rollout of Changes to Cross-Functions
 - Meet with key stakeholders
 - Identify benefits
 - Review issues that the Post-Mortem identified
 - Install feedback process



Seizing Control Part 3: Communicating Results

- Testing-driven Development Lifecycle
 - Rolling up changes in Test/Development Plans
 - Modifying Milestone Criteria
 - Milestone Post-Mortems
 - Use Microsoft Project™ for tracking



Seizing Control Part 4: Examples

- ▶ Real-World Examples of Process Changes
 - Specification feedback
 - Defect discovery: How can it be earlier?
 - Automation re-work too costly
 - Test documentation taking too much time
 - Quality of code
 - Development schedules/process out of sync with milestones



Seizing Control: Conclusions

- ▶ More effective organization
- ▶ Higher quality
- ▶ Less conflict
- ▶ Improved communication
- ▶ Improved morale



Seizing Control: Conclusions

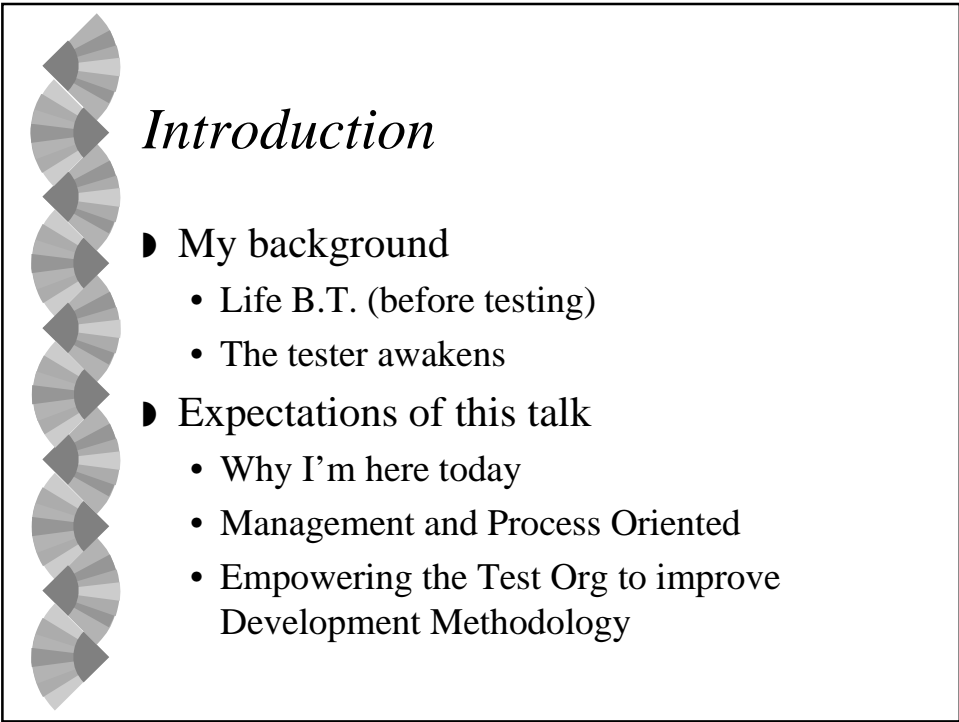
- Success breeds success
 - Peer test groups
 - Cross-functional groups in your company



Thanks for listening!

Questions or additional notes, send
email to:

Nickbo@Microsoft.com





Addressing our differences: Products & Paradigms

- ▶ The labels we live by
 - CMM, ISO, SEI, etc.
 - MIL-Spec
 - Shrink Wrap
- ▶ Common ground
 - We all want a better process
 - Higher quality
 - Accurate Schedules



Why should Testing drive this?

- ▶ Biggest stakeholder?
- ▶ Desire
- ▶ Pragmatism
- ▶ Can any one else do it better?
- ▶ Don't forget the Tester's Mission



Seizing Control: Step 1, Collecting Data

- ▶ In-depth Post-Mortem: Test
 - Various methods to collect data
 - Spend as much energy on your group as critiquing others
 - Important to reach consensus on items in conflict
 - Roll-up big ticket items from Cross-Functions only



Seizing Control Step 1: Collecting Data

- ▶ In-depth Post-Mortem, Cross-Functional
 - Powerful Process to collect both Best/Worst Practices
 - Techniques
 - Moderated Post-Mortems
 - Self-Administrated Post-Mortems
 - Frequency & Follow-up



Seizing Control Part 2: Analysis

- ▶ How can the current Development model hurt test?
 - Identify areas both from Post-Mortem and ones missed
 - How could these change?
 - How would change affect other teams?
 - Would these improve the chances for higher quality and earlier ship date?
 - Refine your "laundry list"



Seizing Control Part 3: Communicating Results

- ▶ Rollout of Changes to your Test/Quality Organization
 - Hash out with your leads/managers
 - Discuss changes with entire team
 - Pitfalls
 - Solving the right problem the wrong way
 - Process change overload
 - Cost/Benefit analysis



Seizing Control Part 3: Communicating Results

- ▶ Rollout of Changes to Cross-Functions
 - Meet with key stakeholders
 - Identify benefits
 - Review issues that the Post-Mortem identified
 - Install feedback process




Seizing Control Part 3: Communicating Results

- ▶ Testing-driven Development Lifecycle
 - Rolling up changes in Test/Development Plans
 - Modifying Milestone Criteria
 - Milestone Post-Mortems
 - Use Microsoft Project™ for tracking




Seizing Control Part 4: Examples

- ▶ Real-World Examples of Process Changes
 - Specification feedback
 - Defect discovery: How can it be earlier?
 - Automation re-work too costly
 - Test documentation taking too much time
 - Quality of code
 - Development schedules/process out of sync with milestones




Seizing Control: Conclusions

- ▶ More effective organization
- ▶ Higher quality
- ▶ Less conflict
- ▶ Improved communication
- ▶ Improved morale



Seizing Control: Conclusions

- ▶ Success breeds success
 - Peer test groups
 - Cross-functional groups in your company



Thanks for listening!

Questions or additional notes, send
email to:

Nickbo@Microsoft.com

Experience-Based Approaches to Process Improvement

Otto Vinter

Brüel & Kjær Sound & Vibration Measurement A/S, DK-2850 Nærum, Denmark

Email: ovinter@bk.dk

Abstract

Software process improvement (SPI) is usually based on well-known models of software process maturity such as the Software Engineering Institute's Capability Maturity Model (CMM) and the European counterpart Bootstrap. This paper reports on alternative approaches to SPI based on knowledge and experience that is already available in the organisation. Rather than a formal comprehensive assessment of the all software development processes in the organisation a "problem diagnosis" is performed. The problem diagnosis approach to SPI aims at identifying the most important (process) issues as perceived by the organisation. Improvement actions are then planned and implemented in close collaboration with the powerful actors in the organisation. The paper focuses on the results of using such problem diagnosis techniques at Brüel & Kjær as an alternative SPI strategy. The paper will report on problems and successes, and relate these results to formal assessments performed in parallel by an external body.

1 Introduction

Software process improvement (SPI) is usually based on well-known models of software process maturity such as the Software Engineering Institute's Capability Maturity Model (CMM) [6] and the European counterpart Bootstrap [2]. The assessment of current practices through the use of such normative models is generally considered as the proper way of identifying and prioritising improvement initiatives.

However, it is also claimed [3] that these models represent a too rigid and limited view of the software development processes and that they do not consider the variety and complexities of software producing organisations.

Furthermore, many organisations have found it very difficult to translate the assessment results into concrete improvement actions. And almost all process improvement persons or groups in organisations at the lower levels of maturity (1-2) have encountered severe resistance to their improvement initiatives from within the organisation. Alternative or complementary approaches to assessments should therefore be considered.

This paper reports on alternative approaches to SPI based on knowledge and experience that is already available in the organisation. Rather than a formal comprehensive assessment of the all software development processes in the organisation a “problem diagnosis” is performed. The problem diagnosis approach to SPI aims at identifying the most important (process) issues as perceived by the organisation. Improvement actions are then planned and implemented in close collaboration with the powerful actors in the organisation.

The paper focuses on the results of using such problem diagnosis techniques at Brüel & Kjær as an alternative SPI strategy. The paper will report on problems and successes, and relate these results to formal assessments performed in parallel by an external body.

The next section (section 2) describes one problem diagnosis approach based on the analysis of defects (error reports) from earlier projects. Two improvement actions were initiated and completed successfully as a result of this approach. In section 3 is reported the findings of the first Bootstrap assessment, which was performed during the implementation of the above-mentioned improvement actions.

Section 4 describes another problem diagnosis approach based on interviews of the leading project managers in the organisation – the powerful actors of a level 2 organisation. Based on the findings from these interviews a number of improvement actions were identified and implemented by the project managers. Improvement actions that directly related to development projects succeeded; those of a more general nature (e.g. organisation-wide) did not.

Section 5 describes the findings of the second Bootstrap assessment, which was performed after the above improvement actions. The findings and recommendations clearly indicate that the improvement actions performed as a result of the problem diagnosis approaches actually helped our organisation to improve on the key practices of a normative model.

Section 6 compares the findings of the different approaches. Section 7 draws the conclusion, that problem diagnosis approaches to SPI seem to be an effective improvement strategy for an organisation at the lower maturity levels (1-2), and a valid alternative to formal assessments according to normative models like CMM and Bootstrap, when an organisation wants to initiate a SPI programme.

2 Defect Analysis

When we first considered improving our processes at Brüel & Kjær, our management did not want to invest in a comprehensive improvement programme. The company had just been through a major reorganisation and downsizing. So rather than starting our process improvement programme with an assessment according to one of the common maturity models, the improvement strategy had to be an experience-driven, incremental process based on the available information in the organisation, and focusing first on the major process issues.

We therefore started by performing root cause analyses of error logs generated during previous development projects. Based on the findings of the analyses we would then introduce improvements in our development process to prevent frequently occurring types of errors. We conducted thorough analyses of bugs reported during development and after release of products. In these analyses we classified bugs according to a taxonomy described by Boris Beizer [1].

The analyses [7][8] showed the need to perform a more systematic unit test of our products. The first improvement action therefore focused on improving our testing process. However, the analyses also showed that the major cause of bugs stemmed from requirements related issues. So when the test improvement action had completed with success, a second improvement action was then undertaken to improve our requirements engineering process.

These improvement actions have been funded by the Commission of the European Communities (CEC) under the ESSI programme: European System and Software Initiative. The title of the test improvement project is: PET - The Prevention of Errors through Experience-Driven Test Efforts (ESSI project no. 10438) [7][8]. The title of the requirements engineering project is: PRIDE - A Methodology for Preventing Requirements Issues from Becoming Defects (ESSI project no. 21167) [9].

2.1 The Test Improvement Project

The software quality of our company was felt to be unsatisfactory. Too many products were shipped with bugs. It was the general opinion that this was caused by a lack of testing by the developers before release.

It was therefore decided to conduct a process improvement experiment to find ways to improve the testing process. The project was titled: The Prevention of Errors through Experience-Driven Test Efforts (PET) [7][8].

The problem reports were analysed and bugs in them categorised using Boris Beizer's taxonomy [1]. We found that bugs in embedded real-time software follow the same pattern as other types of software. We found that the major cause of bugs reported (36%) are directly related to requirements, or can be derived from problems with requirements. The second largest cause of bugs (22%) stems from lack of systematic unit testing.

The techniques selected to improve unit testing were: Static and dynamic analysis. Tools were installed to support these techniques. We experienced a 46% improvement in testing efficiency (bugs found per person hour) and we raised the branch coverage of all units to above 85%.

An improved (production) version of the baseline product was then released and tracked for the same number of weeks we had measured on the existing (trial) version after its release, so that we were able to evaluate the effect of the experiment on problem reports.

The team received 75% fewer error reports than for the trial-release version of the product. Of those error reports 70% were found to be related to requirements e.g. to bugs that could not have been found through static and dynamic analysis. This once more confirmed the need for us to improve the requirements process.

In spite of these remarkable results the use of static and dynamic analysis never spread throughout the organisation. Some project managers ignored the results. Others started to work with the techniques, but stopped when time pressure built up. Those who continued, released products with remarkably fewer bugs.

2.2 The Requirements Engineering Improvement Project

In the second improvement action we performed a closer analysis of requirements related bugs in order to find and introduce effective prevention techniques in our requirements engineering process. The project was titled: A Methodology for Preventing Requirement Issues from Becoming Defects (PRIDE) [9].

From the analysis of requirements related bugs we found that requirements issues are not what is expected from the literature. Usability issues dominate (64%). Problems with understanding and co-operating with 3rd party software packages and circumventing their errors are also very frequent (28%). Functionality issues that we (and others) originally thought were the major requirements problems only represent a smaller part (22%). Other issues account for 13%. The sum of these figures adds up to more than 100% because one bug may involve more than one issue.

This result had an impact on our methodology. We focused on usability problems, and early verification and validation techniques, rather than correctness, and completeness of requirements documents.

We therefore introduced the following techniques on a real-life project:

- *Scenarios*
Relate demands to use situations. Describe the essential tasks in each scenario.
- *Navigational Prototype Usability Test, Daily Tasks*
Check that the users are able to use the system for daily tasks based on a navigational prototype of the user interface.

We found an overall reduction in error reports of 27%. We saw a 72% reduction in usability issues per new screen, and a 3 times increase in productivity in the design and development of the user interface.

What was also surprising was that not only did we experience a reduction in bugs related to requirements issues, we also found a reduction in other bug categories. The derived effect on other types of bugs than the requirements related can be explained by the fact that most of the developers achieved a deep understanding of the domain in which the product was going to be used from describing use situations (scenarios) and taking part in the usability tests. This invariably leads to a reduced uncertainty and indecision among the developers on what features to include and how they should be implemented and work. In the previous project the new screens were constantly subject to change all through to the end of the project.

However, the impact of these techniques on the perceived quality of the released product is even greater than the prevention of bugs. Describing use situations (scenarios) enabled the team at a very early stage in the requirements engineering process to capture the most important demands seen from a user/customer perspective. The developers therefore got a very clear vision of the product before the requirements were fixed. The subsequent usability tests on very early prototypes verified that the concepts derived from the descriptions of use situations (scenarios) still matched the users' needs and could be readily understood by them in their daily use situations.

The product has now been on the market for more than 18 months and it steadily sells more than twice the number of copies than the product we have compared it to. This is in spite of the fact that it is aimed at a much smaller market niche, and that the price of the new product is much higher.

In contrast to the test techniques, the interest among project managers to adopt the scenario and usability techniques has been much higher. This may be because developers much rather will work with requirements than with test.

3 Bootstrap Assessment

When the first results of the improvement actions based on defect analysis had materialised, the management of our company could be convinced that a more formal assessment of our software development processes should be performed in order to further the improvement programme.

A Bootstrap assessment was performed using the Danish company DELTA as assessors. Four projects and the software development management were interviewed by the external assessors. The overall result of the assessment was that we were at level 2.25.

The recommendations from the assessors pointed out weaknesses in the following areas:

- **Development Model**
The shift in focus from a primarily hardware driven development to software had to be more focused. The assessors recommended to introduce a specific life-cycle for software development.
- **Process Descriptions**
Introduce the formal as well as informal improvement actions of the software processes in the quality management system.
- **Unit and Integration Testing**
The assessors commented that the improvements we had achieved under the test improvement experiment (PET) needed to be applied on a wider scale in the company.
- **Configuration Management**
Again the assessors commented on the need for a more uniform way of performing configuration management, change control, and planning.
- **Requirements**
The assessors were aware of the ongoing requirements engineering experiment (PRIDE) and commented on the need for a description of the process.
- **Project Management**
Improve planning, estimating procedures, introduction of time and resource usage, closer monitoring of project progress.

The recommendations above clearly show that a Bootstrap assessment has a much wider perspective of the software development process than defect analysis. Defect analysis primarily highlights “hot-spots” in the development process. However, the testing and requirements “hot-spots” that we found through our defect analysis were also found through the assessment.

Due to the strong project manager culture of our company, which is typical of a level 2 maturity, the recommendations from the assessment were never turned into improvement actions. Top management had stated their commitment to follow up on the assessment recommendations, but in the end they left it to the project managers themselves to find and introduce improvement actions on their individual projects. And the process improvement group did neither have the resources nor the “power” to be able to introduce new activities on a wider scale.

With no “pressure” from the top, and projects running late, it is no wonder that the project managers chose to concentrate on their day-to-day problems of getting products out of the door. Consequently, no improvement actions were started as a result of the assessment. This is characteristic of an organisation of level 1-2 maturity.

4 Project Manager Involvement

At this point it was evident to us that we were effective in defining and introducing new and improved processes on individual development projects. However, the diffusion and adoption of the techniques on an organisation wide scale did not happen by itself. The failure of the Bootstrap assessment to spur new improvement initiatives among the project managers themselves made it evident to us that we had to involve these powerful organisational actors directly in order to improve on a broader scale.

For this to happen we needed to increase our resources for process improvements, and we succeeded in convincing our management to let Brüel & Kjær participate in the Center for Software Process Improvement [5], which is partly funded by the Danish government. The participation gave us access to a number of researchers and consultants and the resulting joint effort became the basis for spreading improvements on a wider scale in the organisation.

We involved the project managers directly by performing a new type of problem diagnosis. We wanted to analyse which (process) issues were perceived as important by them. The most common problems would then form the basis for our next improvement activities, and this - we hoped - would enable an easier diffusion and adoption across the projects.

The problem diagnosis technique [4] we used was to conduct a series of structured interviews with each of the leading project managers where they were asked about which type of problems in their development projects they felt were the most serious. Seven project managers were interviewed and detailed minutes were recorded.

From these minutes we could compose a list of problem areas that the project managers found to influence their projects most. The problems perceived by the project managers correlated very well, so a consensus could be reached quite easily:

- Software Development Model
The present ISO9001 registered waterfall model was deemed unsatisfactory for efficient software development. A new model based on iterations, e.g. through experimental prototypes, was asked for. Risk management was also mentioned as an important element.
- Requirements
The project managers were aware of the ongoing requirements engineering improvement action (PRIDE) and requested improvements in this area.

- Project Monitoring
Better estimating procedures, follow up, and progress evaluations.
- Project Conclusion
Configuration management, release criteria, and testing.
- Reuse
Actually this item did not appear in the interviews, but the organisational changes that took place at the time of the evaluation centred on establishing a group responsible for reuse. So the process improvement group added reuse to the list.

The problems perceived by the project managers clearly resemble the recommendations from the Bootstrap assessment. Once again we see a much wider perspective of the software development process than defect analysis. However, the testing and requirements “hot-spots” that we found through our defect analysis were also found through this evaluation.

The project managers were presented with the findings at a workshop where top management also was present. They were each asked to select a topic from the list that they felt most natural to work with on their present (or up-coming project). The process improvement group and the researchers from the Center for Software Process Improvement then established support groups that would train, mentor, and follow these projects.

Three project managers chose to work with implementing the requirements engineering techniques from PRIDE (section 2.2). Two chose to work with new development models. One chose to work with reuse and one with project conclusion. Finally the R&D manager wanted to contribute by improving project monitoring.

These new improvement actions were performed headed by the project managers with mentoring and support from the process improvement group.

The reuse improvement project was initiated, but the group responsible for this never received the necessary resources and support to get the project going. This is in fact quite natural since the item was added by the improvement group and not perceived as a problem by the project managers. We are a level 2 company, and reuse is an organisation wide activity (level 3).

Similarly the project monitoring improvement project did not succeed. The R&D manager acquired and implemented a tool to support estimation and monitoring activities. But since he did not press the project managers to use the tool, which by the way also had quality problems, the project managers to a very large extent simply ignored the tool. The improvement group tried to establish a support group, which should assist in getting the project managers to accept the tool, but the R&D manager felt that he was too busy to take part in such an activity. The failure of this improvement project is also quite natural. A tool by itself never solves a process problem, you need methodology and organisational support, and this was not achievable at our level of maturity.

The project conclusion improvement project concentrated on testing. It did not directly use the tools and techniques from PET (section 2.1). However, the philosophy from PET that a project should achieve a certain level of testing before release was permanently in the minds of the people working on the project. A number of persons were assigned to testing the products, and they managed to

achieve the necessary credibility from the developers and project managers, so that the products were not released until accepted by them.

Finally the three projects that decided to work with the requirements techniques, and the two projects that wanted to work with development models were successful. There seemed to be very little resistance to the improvement actions recommended by the support group. In fact there was great enthusiasm in the teams for both the development model experiments and requirements engineering techniques.

The conclusion that can be drawn from these improvement actions is that only those improvements will be successful that are perceived by the project managers as solving their problems. Neither the introduction of tools nor organisation wide actions will be effective in an organisation at maturity level 1-2 unless they align with the “power structure”.

5 Did We Improve Our Maturity?

After the improvement actions mentioned in the previous sections had been completed a second Bootstrap assessment was performed in order to check on the results and further the improvement programme.

The Bootstrap assessment was performed by the same assessors from DELTA. Four projects and the software development management were interviewed by the external assessors. The overall result of the assessment was still level 2.25.

This immediately seems like no improvement. However, when you study the detailed results that lead to this number, you can see that the first assessment result was only a little more than halfway between 2.0 and 2.25 (and the final result therefore achieved through rounding), the second is straight at the level.

More important, however, than this number was the changes in recommendations made by the assessors. This time they pointed out weaknesses in the following areas:

- **Configuration Management**
The assessors commented on the need for a more uniform way of performing configuration management, change control, and planning.
- **Project Management**
Improve planning, estimating procedures, introduction of time and resource usage, closer monitoring of project progress.
- **Management Responsibility**
Improve status reporting, roles and responsibilities, common policies and strategies.
- **Architectural Design**
Introduce interface design methods and standards.
- **Process Measurements**
Introduce measurement programme and procedures.

- (Unit and Integration Testing)
The assessors acknowledge that the present level of testing seems adequate for the quality of the products in the market.

These recommendations clearly show that those improvement actions that the project managers have undertaken w.r.t. development model and requirements are no longer on the list. The testing recommendation has been moved to the bottom position of the list, and the assessors acknowledge that the testing performed seems to be adequate for products like ours, even though it is not performed fully according to the maturity model.

The recommendations from the previous assessment on configuration management and project management that we had either ignored in our improvement actions (configuration management) or not succeeded in getting to work (project management) are still on the list and now at the top of the list.

Based on the new recommendations from the assessors, two further improvement actions have been initiated. One improvement action addressing configuration management, and one action attempting to revitalise the acceptance and use of the estimation and monitoring tool, this time focusing on project feed-back and management reporting, e.g. addressing the organisation around the tool. It remains to be seen whether we will be successful in implementing these actions.

We also intend to repeat the problem diagnosis by performing another round of interviews with the project managers in order to evaluate the changes in the (process) issues that they perceive as important to them.

6 Comparison of Approaches

The recommendations from the different approaches to SPI that we have performed show a great deal of overlap though they sometimes use different words.

| Recommendation | Defect Analysis (section 2) | Bootstrap Assessment (section 3) | Project Mgr. Interviews (section 4) |
|---|--------------------------------|--|---|
| Development Model - iterations - risk management | x | x | x x |
| Requirements | x | x | x |
| Project Monitoring - estimation - time & resource usage - monitor progress | | x x x | x x |
| Project Conclusion - configuration mgmt. - testing - release criteria | x x | x x | x x x |
| Reuse | | | x |
| Process Descriptions | | x | |

What is important, however, is the fact that the findings from the defect analysis approach are recommended by the more general evaluations. This means that a problem diagnosis approach to SPI based on the available error logs in the organisation is a valid approach to initiate a process improvement programme. And we have seen that improvement actions based on the defect analysis approach actually lead to improvements that changed the recommendations in a formal Bootstrap assessment.

The majority of the assessment recommendations are also found by the approach based on project manager interviews. This means that a problem diagnosis approach to SPI based on the (process) issues perceived by the project managers is also a valid approach to initiate or sustain a process improvement programme. And we have seen that improvement actions, which are adopted by the project managers, actually lead to improvements that changed the recommendations in a formal Bootstrap assessment.

7 Conclusions

From the above comparison of approaches and the results of our improvement actions, we conclude that problem diagnosis approaches to SPI based on already available experience in the organisation are effective and valid alternatives to formal assessments for organisations at maturity level 1-2.

Normative models for SPI like CMM and Bootstrap should not be discarded because of this. They represent a comprehensive framework for improvement actions and have established a way of measuring (assessing) whether improvement actions have been successful within an organisation.

However, their basic principle of establishing levels of maturity, and consequently the recommendation for organisations to improve from one level to the next until the top level is reached, seems to be very difficult for organisations to accomplish. This may discourage many organisations from improving their processes at all.

Alternative approaches to SPI like the ones we have employed tend to be hot-spot solutions to very specific problems. However, they have a high motivational effect because of the immediate quick wins they bring. This will encourage the organisation to pursue a continuous line of improvement actions, and as we have seen, completely in sync with the key process improvement areas of the normative models.

Irrespective of the SPI approach, it is our experience that in order to achieve a widespread adoption of process improvements in an organisation, it is necessary that the improvement actions are defined by or co-ordinated with the powerful actors in the organisation, which at a level 2 maturity are the project managers. Recommendations from a Bootstrap assessment or dedicated improvement actions will only be effective if they are aligned with the “power structure” of the organisation.

8 References

- [1] Beizer B., *Software Testing Techniques. Second Edition*, Van Nostrand Reinhold New York, 1990.
- [2] Bicego A., Khurana M., Kuvaja P., *BOOTSTRAP 3.0: Software Process Assessment Methodology*, Proceedings of the SQM'98, 1998.

- [3] Bollinger, T. B. and McGowan, C., *A Critical Look at Software Capability Evaluations*, IEEE Software, Vol. 8, No. 4, pp. 25-41, 1991.
- [4] Iversen, J., Nielsen, P. A., and Nørbjerg, J., *Problem Diagnosis in Software Process Improvement*, Proceedings of the IFIP WG8.2 & WG8.6 Working Conference, Helsinki, Finland, 1998. (<http://www.bi.no/dep2/infomgt/wg82-86/proceedings/iversen.pdf>)
- [5] Mathiassen L., *Beskrivelse af forskningsprojekt om: Softwareprocesforbedring*, Aalborg Universitet, DK-9220 Aalborg Øst, Denmark, 1996.
- [6] Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V., *Capability Maturity Model for Software, Version 1.1*, 93-TR-024, Software Engineering Institute, Pittsburgh, Pennsylvania, 1993.
- [7] Vinter O., Poulsen P.-M., Nissen K., Thomsen J.M., *The Prevention of Errors through Experience-Driven Test Efforts. ESSI Project 10438. Final Report*, Brüel & Kjær A/S, DK-2850 Nærum, Denmark, 1996. (<http://www.esi.es/ESSI/Reports/All/10438>).
- [8] Vinter O., Poulsen P.-M., Nissen K., Thomsen J.M., Andersen O., *The Prevention of Errors through Experience-Driven Test Efforts*, DLT Report D-259, DELTA, DK-2970 Hørsholm, Denmark, 1996.
- [9] Vinter O., Lauesen S., Pries-Heje J., *A Methodology for Preventing Requirements Issues from Becoming Defects. ESSI Project 21167. Final Report*, Brüel & Kjær Sound & Vibration Measurement A/S, DK-2850 Nærum, Denmark, 1999. (<http://www.esi.es/ESSI/Reports/All/21167>)

Appendix A: CV of Author

Otto Vinter is managing a software technology and process improvement group at Brüel & Kjær responsible for projects to improve the software development process. He has been active in defining software engineering standards, procedures, and methods to be employed at Brüel & Kjær. He has also been the driving force in the company's transition from procedural programming to Object-Oriented development.

He has managed software development projects for almost 30 years; with Brüel & Kjær from 1986, before that with the Danish branch of Control Data Corporation, and with Regnecentralen.

The author received his Masters Degree in Computer Science from the Danish Technical University in 1968. He is an associate teacher for BSc. level education in Computer Science, is an active participant in Danish software knowledge exchange groups, performs mentoring activities for clients, and is on the programme committee of several international conferences.

Appendix B: The Company

Brüel & Kjær is a leading manufacturer of high-precision measurement instruments. Brüel & Kjær develops high-precision electronic instruments for sound and vibration measurement applications. The company is headquartered in Denmark, but the majority of the products are sold through subsidiaries around the world. In the past most of the products were based on embedded real-time software, but now PC applications prevail.

Brüel & Kjær Sound & Vibration Measurement A/S
Skodsborgvej 307, DK-2850 Nærum, Denmark
Tel: +45 4580 0500, Fax: +45 4580 1405

Quality Week Europe '99

Experience-Based Approaches to Process Improvement

-

Otto Vinter

Manager Software Technology and Process Improvement

Tel: +45 4580 0500, Fax: +45 4580 1405

ovinter@bk.dk

&

Software Engineering Mentor

Tel/Fax: +45 4399 2662, Mobile: +45 4045 0771

vinter@inet.uni2.dk <http://inet.uni2.dk/~vinter>

Normative Models for SPI

CMM

BOOTSTRAP

SPICE (ISO15504)

Defined comprehensive process

Levels of maturity (capability)

Key process areas for each level

Assessed by certified assessors

Fundamentals in Normative Models

| Level | Characteristics | Principal Characteristics | Result |
|-----------------|---|--|------------------------|
| 5 OPTIMISING | feedback — the process includes feedback | Continuous evolution & continuous process improvement | Productivity & Quality |
| 4 MANAGED | quantitative — the process is measured | Technological replacement Problem prevention | |
| 3 DEFINED | qualitative — the process is institutionalised | Process measurement Process analysis Quantitative quality plans | |
| 2 REPEATABLE | intuitive — the process is person-dependent but it is managed | Training Systematic review & testing Standards | |
| 1 INITIAL | ad hoc — chaotic | Project management Planning Configuration control Quality assurance | |

Software Process Assessment & Improvement (The BOOTSTRAP Approach)
ISBN 0-631-19663-3

Problems with Normative Models

Abstract model
Assessment by external body
Costs money
Organisational focus
Points out KPAs to be improved
Little help on precisely what to do
Raises a lot of expectations

Alternative Approaches to SPI

Experience-based improvement actions

- Analyse problems from previous projects to extract knowledge on frequently occurring problems
- Change the development process through the use of an optimum set of methods and tools available to prevent these problems from reappearing
- Measure the impact of the changes in a real-life development project
- Diffuse the results to the rest of the organisation

Characteristics of Alternative Approaches

Experience-based
No specific model
Hot-spot driven
Focus on prevention
One issue at a time (incremental)
Piloting at project level
Evolve rather than define (feed-back)
Fits CMM level 1-2 cultures
(where most of us are)

Examples of Alternative Approaches

Analysis of

- defects
- progress reports, etc.

Structured/Selective interviews

- project managers
- project members
- customers, etc.

Goal-Question-Metric paradigm (GQM)

Some frameworks for alternative approaches:

- Experience Factory (V. Basili e.a.)
- Product Process Dependency Models (PROFES: www.ele.vtt.fi/profes)
- but primarily: you must find your own way

1st Action: The Prevention of Errors through Experience-driven Test Efforts (PET)

Results of the analysis of error logs

- no special bug class dominates embedded software development
- requirements problems, and requirements related problems, are the prime bug cause (>36%)
- problems due to lack of systematic unit testing is the second largest bug cause (22%)

Action: Introduction of static/dynamic analysis

- 75% Reduction in bugs after release
 - 46% Improvement in testing efficiency
 - 70% Requirements bugs after release
- => Increased focus on the requirements process

Funded by CEC. Final Report: <http://www.esi.es/ESSI/Reports/All/10438>

2nd Action: A Methodology for Preventing Requirements Issues from Becoming Defects (PRIDE)

Results of the analysis of error logs

- Requirements related bugs 51%
- Usability issues dominate 64%
- External software (3rd party & MS products) 28%

Action: Introduce requirements techniques

- Use Situations (Scenarios)
 - Relate demands to use situations. Describe tasks for each scenario.
- Usability Test, Daily Tasks, Navigational Prototype
 - Check that the users are able to use the system for daily tasks, based on a navigational prototype of the user interface.

Funded by CEC. Final Report: <http://www.esi.es/ESSI/Reports/All/21167>

Results of the 2nd Improvement Action

Quantitative

- Developers were almost 3 times as productive in the development of the user interface
- Usability issues per new screen reduced by: 72 %
- Total reduction in error reports: 27 %

Qualitative

- User interaction with the product was totally changed as a result of the usability tests
- Selling steadily more than twice as many
- Product was released in December 1997 so orders could be shipped before the end of the financial year

3rd Action: Project Manager Interviews

Problem diagnosis:

- Slow diffusion and adoption of results
- 1st Bootstrap assessment had no effect
- Project managers were the powerful actors (level 2)

Action: Involve the project managers

- interview them on their (process) problems
- classify the (process) issues
- present the major common issues for them (workshop)
- let each select an issue to improve
- train, coach, and support them and their teams
- collect experience and results

Results of the 3rd Improvement Action

Seven managers interviewed

- problems perceived by them correlated well

Workshop on the major issues

- three chose requirements techniques
- two chose new iterative development models
- one chose project conclusion (test and release)
- R&D manager chose project monitoring

Follow-up by SPI support group

- Introduce and train the team on the issue
- coach and support on request (at least monthly)
- improvement actions performed with enthusiasm
- project related actions completed successfully

Comparison of Recommendations

| Recommendation | Defect Analysis | 1 st Bootstrap Assessment | Project Mgr. Interviews | 2 nd Bootstrap Assessment |
|-----------------------------|-----------------|--------------------------------------|-------------------------|--------------------------------------|
| Development Model | | | | |
| - iterations | x | x | x | √ |
| - risk management | | | x | |
| Requirements | x | x | x | √ |
| Project Monitoring | | | | |
| - estimation | | x | x | x |
| - time & resource usage | | x | | x |
| - monitor progress | | x | x | x |
| Project Conclusion | | | | |
| - configuration mgmt. | | x | x | x |
| - testing | x | x | x | (√) |
| - release criteria | x | | x | |
| Reuse | | | x | |
| Process Descriptions | | x | | √ |

Problem Diagnosis Results

Defect analysis from error logs

- has established a basic process for testing and release
- has improved our requirements process and products

Improvements from interviews

- successful diffusion and adoption of previous improvement actions
- established a new software development model

Impacts on maturity

- development model, test, and requirement issues are no longer on the Bootstrap recommendation list

In Conclusion

Problem diagnosis

- is a simple and effective way to find problems in the software development process
- good starting point for process improvement programmes in companies
- step-wise improvements with quick wins
- changes assessment recommendations

However, normative models are needed

- comprehensive framework
 - KPAs are important, forget the levels
- established assessments
 - effect of improvements

Formal Methods in Robotics: Fault Tree Based Verification

Axel Lankenau and Oliver Meyer

Bremen Institute of Safe Systems
TZI, University of Bremen
P.O. Box 330440, D-28334 Bremen
{alone, emm}@tzi.de
phone: +49-421-218-{4684, 3337}
FAX: +49-421-218-3054

Abstract. The intention of this paper is to emphasize the importance of employing formal methods for the design of robotic systems. After a brief survey of current research in this area, a set of requirements is discussed that formal development process should fulfil. As an extended example, a general verification approach for reactive systems is described in detail. It is based on a CSP specification of a fault tree that observes the behaviour of the target system. A template for the modelling of fault tree leaves and nodes is given, and it is instantiated by a “real-world” application taken from the field of mobile robotics.

Keywords: Safety in Robotics, Formal Verification, Fault Tree Analysis

Target Audience: Technical

Basis: Academic Research

1 Introduction

When considering systems engineering tasks in the context of avionics or railway interlocking systems, the use of formal methods in the design process is becoming state of the art. Due to the fact that a malfunction of such systems may cause severe harm to human beings or result in other catastrophic consequences, these systems are referred to as *safety-critical systems* [27, 16]. Formal methods are used to formally specify the safety requirements of these systems, to define the physical model of the system and its environment. They facilitate the verification, validation and test (VVT) processes and are absolutely necessary if these processes are to be automated.

As the potential damage robot actions can cause to human beings increases with the spatial proximity in which man and machine operate, service and especially rehabilitation robots have to be classified as safety-critical systems.

This point of view represents an extension to a common understanding in the robotics community that a *reliable* robot is a *dependable* robot (for definitions cf. [14]). This statement may be true for exploration robots which operate in

volcanoes or on Mars, but it is an inadequate simplification of matters with respect to those robots that work in the vicinity of men. For them, the notion *dependable* additionally comprises the properties *available* and *safe*.

There exists a lot of knowledge how safety-critical systems such as nuclear power plants or airplanes have to be built. Various requirements engineering methods, a huge variety of specification languages, many verification approaches as well as test tools have been developed. This paper intends to specify the demands that applications in robotics impose on a formal method to be employed in this domain. For the first time, a method which comprises hazard analysis, the formal specification of safety requirements and their subsequent verification is applied to a service robotics system.

2 Formal Methods in Robotics — A Brief Survey

In contrast to other application fields such as avionics, the use of formal methods is not yet very widespread in the new research areas of service and rehabilitation robotics. This will change in the future for two reasons: on the one hand, most robotic systems are “interestingly complex”, i.e. they are more demanding than academic toy examples but are not too extensive to handle. On the other hand, it is commonly agreed that robots such as human power extenders [10] or intelligent wheelchairs [2, 19, 20] should be treated as safety-critical systems.

However, up to now there are only a few research groups which concentrate on the topic of using formal methods to ensure safety properties for (service) robots.

At Lancaster University various approaches to safety analysis were examined. In [24, 26] the concept of a “safety manager” was introduced. An autonomous excavator serves as an application example in these papers. They focus on system analysis topics (development of the hazard analysis method CLASH), and do not cover the verification and validation process [23, 25, 11].

Zhang employed a far more formal approach by developing a semantic model for dynamical systems, the so-called “constraint nets” (cf. [30]). In addition, she proposes a formal verification concept.

The research group “Robots for Hazardous Environments” at Rice University deals with fault-tolerant robot architectures (e.g., cf. [15]). This work aims at improving the reliability of robots through software engineering methods such as fault-tree analysis or risk analysis [6]. Formal verification is not covered.

3 Requirements for Formal Methods in Robotics

Due to the complexity of robotic applications mentioned, the employed formal methods should cover several different problem domains.

Robotics applications are always *hybrid real-time systems*, e.g. industrial robots have to control the angles of joints, mobile robots deal with speeds and braking distances etc. Therefore, the employed formal methods for specification,

validation, verification and test have to cope with the combination of discrete events and continuous values and their effects on each other. Furthermore, it must be possible to specify and analyse hard real-time properties. Most tool-supported formal methods used today are only capable of handling discrete systems.

Robots, like all reactive systems, are only able to operate correctly if the environments they are operating in fulfil certain conditions. The specifications of these conditions are the basis for the VVT phases. It is desirable that the formal methods not only allow to take these requirements on the environment into account but provide methods to deduce them during the specification phase of the system.

The complexity of robotics systems requires hierarchical methods which allow the stepwise refinement from abstract to more concrete properties. Otherwise it would be almost impossible to check the completeness of specifications or correctness proofs. Furthermore, hierarchical methods are usually compositional. This makes it possible to easily re-use parts of specifications for similar problem domains. Since the high levels are only describing abstract properties of robots in general without details about the specific hardware or the intended implementation, their re-use reduces the necessary effort to argue about the completeness of specifications for new projects. Besides, compositionality of the specification method might enable the partial re-use of proofs or proof scripts for these developments.

In order to simplify the handling of the previous aspects it is essential to support the applied formal methods by tools which cover the whole development cycle. It should be possible to use the results of the specification phase for the whole VVT process without having to rewrite them in different languages or formalisms.

The formal approach applied in this paper focusses on the aspects of (interval based) hard real-time properties, deduction of environment specifications and compositionality/re-usability. The integration of *general* real-time requirements as well as complete support for hybrid systems is part of our current work.

4 Fault-Tree Based Verification in CSP — An Extended Example

In the sequel, a new verification method is introduced and instantiated by a service robotics example. It is based on a fault tree documenting the result of a structured hazard analysis. The fault tree specifies threats the system is exposed to and makes it possible to consistently define safety requirements that the operational system must never violate. By modelling the fault tree in CSP it becomes the reference specification of undesired behaviour the system is verified against. Verification is accomplished by model checking to ensure that the top hazard specified within the fault tree hierarchy never occurs.

On the other hand, the same fault tree may serve as a basis for test evaluation if an already implemented system is to be automatically tested. In this case, the

I/O behaviour of the system and the environment can be modelled as a sequence of events, and thus becomes able to trigger the implemented fault tree leaves, causing the propagation of faults through the hierarchy of hazards.

As the example presented below is part of a robotics application (the Bremen Autonomous Wheelchair, see [13]), another subtle advantage of the fault tree based verification is worth mentioning. In robotics, most research projects confine themselves to do mere existence proofs, i.e. it is only shown that a certain robot performs a specific task. In most cases, there is no detailed information given with respect to the circumstances and the environmental conditions under which the experiments were carried out. As indicated above, an additional outcome of the verification method described here is a detailed set of hypotheses specifying the requirements the environment has to satisfy in order to allow the robot to behave as intended.

Besides providing the benefits “one reference document for verification and automated testing (cf. section 5.2)” and “specification of the environment”, the approach presented here may be easily adapted to various other problem domains in which embedded controllers are developed.

4.1 Fault Tree Based Verification

Fault tree analysis allows a hierarchically structured and complete derivation of the hazards of a system that has to satisfy safety properties (cf. [28], [7]). It leads to a fault tree that gradually links abstract to more specific threats. The leaves of the tree describe the basic hazards that have to be taken into consideration in system design.

In this paper a graphical representation of fault trees is used that is based on the “alternative symbols” from [9]. Leaves or nodes which are on the same level are a refinement of the higher hazard. The higher hazard occurs if and only if all lower hazards occur (logical AND) or if one of the lower hazards occurs (logical OR), respectively. Note that the occurrence of a hazard does not necessarily mean that the corresponding catastrophe happens, it only means that it is possible to happen. In order to be able to refer to leaves (depicted as circles) and nodes (depicted either as rectangles or as lozenges), they are hierarchically numbered. Besides, we extended the fault tree symbols by a combination of “triangle” and “lozenge” called “external hazard”. It represents a hazard which has to be averted by external safety mechanisms and thus does not have to be considered in detail during the system design. The environment requirements are deduced from the set of external hazards.

Consider the small example shown in Fig. 1. In this fault tree hazard **A** occurs if and only if the lower hazards **B** or **C** occur. Accordingly, hazard **B** occurs whenever basic hazard **D** or **E** occurs. Furthermore, hazard **C** occurs if and only if the basic hazard **F** and **G** exist simultaneously, but hazard **G** is an external hazard which is averted by the environment. This leads to an implementation where the developed system may only operate correctly in environments providing the necessary safety mechanisms to avoid the corresponding hazard **G**.

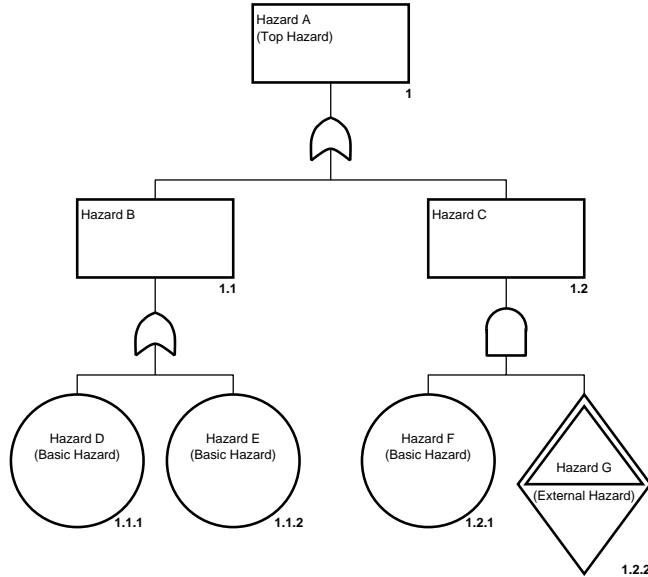


Fig. 1. A simple fault tree example

Deriving Safety Requirements from Fault Trees. When using fault trees as a decomposition of the threat space of a system, the usual procedure is to derive *safety requirements*. These are simply negations of the basic hazards. Ensuring that these safety requirements hold is equivalent to ensuring the non-occurrence of the corresponding hazard. Note that it is sufficient for AND-connected hazards if only one of the safety requirements holds to prevent the occurrence of the higher hazard. Besides, it may sometimes be possible to avert higher level hazards and thus one does not have to bother with the more specific hazards in these cases.

The next step is to develop *safety mechanisms* which ensure that the corresponding safety requirements always hold. This has to be realized by taking into account the fault tree hierarchy. It is the task of the safety mechanisms to prevent the environment and a naive realization of the system from performing undesired sequences of events, i.e. it has to block those sequences that may lead to a catastrophic system state. In the sequel, *equipment under control (EUC)* is referred to as representing a simple implementation of the desired functionality without taking into consideration the safety requirements, whereas *system* means the combination of EUC and safety mechanisms.

Since the development of the safety mechanisms cannot be done automatically, it may be erroneous or incomplete. Therefore, a verification is necessary to guarantee that the safety mechanisms are correct with respect to the acceptable

faults¹, i.e. to guarantee that the top hazard in the fault tree can never occur (see Fig. 2). Our approach to perform this verification is outlined in the following section.

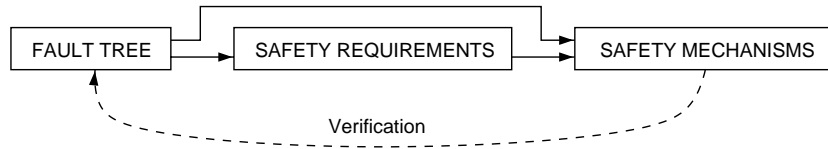


Fig. 2. Dependencies between fault tree, safety requirements and safety mechanisms. The solid arrows represent a “is derived from”-relation. The process for the derivation of safety mechanisms may be erroneous and therefore a verification is needed.

It is worth mentioning that the approach chosen does not ensure necessarily safety with respect to *all* possible hazards, but only with respect to those specified in the initial reference document. Here, this document is the fault tree. As its development supports complete coverage of the threat space, it is an adequate means to get the first specification as complete as possible.

Verification Approach. Since the fault tree describes the undesired system behaviour it can be used to verify the correctness of the derived safety mechanisms. It has to be guaranteed that the system can detect and/or treat the basic hazards appropriately to avoid the occurrence of the top hazard in the fault tree.

The basic idea is to use an implementation of the fault tree that runs in parallel with the system and the environment and observes their behaviour. Here, *implementation* of the fault tree refers to a formal specification of the fault tree that can be symbolically executed during the verification process. Whenever a fault leads to a basic hazard it is propagated up the tree according to the specified hierarchy and logic connections between related leaves and nodes. If the propagation continues up to the root the system failed.

In order to ensure that the system *always* behaves correctly it is necessary to examine all possible interleavings of the system components and the environment. This can be done by using a model checker on a formal specification of the safety mechanisms, the environment and the fault tree. Alternatively, classical verification methods could be employed, but they lack the ability to completely automate the verification process.

Both the environment specification and the safety mechanism specifications can trigger exceptions which correspond to the occurrence of a basic hazard.

¹ If hazards are AND-connected the occurrence of single faults can be accepted because it does not lead to the higher level hazard.

These signals are called “artificial events” since they can often be provided by the model of the environment only but not by the real environment. For example, consider a sonar sensor: A model could nondeterministically decide to measure a wrong distance and signal it to the fault tree but a real sensor has no means of detecting it. Besides, it is possible that the safety mechanisms themselves are incorrect. They might introduce new threats and therefore they, too, must be able to trigger artificial events (see Fig. 3).

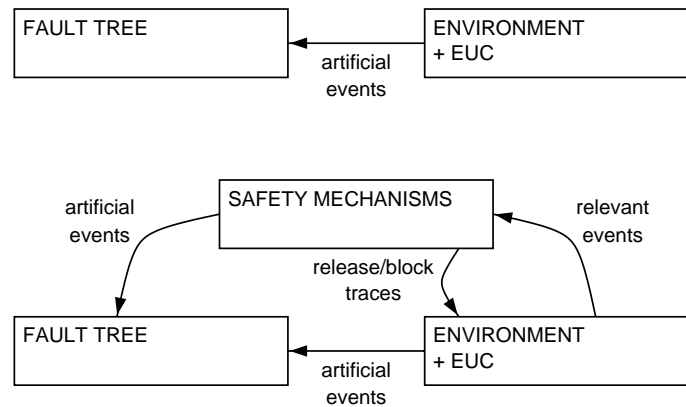


Fig. 3. The basic idea of the fault tree based verification approach. The environment, the equipment under control (EUC) and the safety mechanisms can trigger artificial events which are observed by the fault tree. In the figure, the upper composition without safety mechanisms can be used to analyse the system behaviour and to demonstrate that the safety mechanisms are necessary. In contrast, the lower composition is employed to verify the non-occurrence of the top hazard.

4.2 Fault Tree Implementation in CSP

When taking into account that the verification of the safety properties mentioned above is carried out by model checking, it becomes obvious that a formal specification of the fault tree is needed. It is especially important to model the propagation of fault-triggers from the leaves to the root of the tree. As this inherently includes the modelling of communicating processes, the formal specification language CSP (cf. [8] and [21] for syntactic extensions) is used.

An additional reason to choose CSP was the model checking tool FDR (cf. [3]) which allows verifying various refinement relations between specified processes. The template specifications of a leaf, an OR-node and an AND-node are rep-

resented in FDR syntax in Figs. 4, 5 and 6, respectively. Thus, it is possible to directly re-use the depicted code segments in other applications.

The basic idea used here is a kind of discrete time slicing. It is assumed that the system works in a loop lasting, say, t time units. During this period of time, the environment and the system may cause faults, i.e. trigger those artificial events the leaves of the fault tree engage in. Meanwhile, the implementation of the fault tree merely observes, i.e. its leaves never refuse the artificial events triggered by the system or the environment. This behaviour is absolutely essential, because the fault tree is a “virtual observer” that must not have any influence on the system or the environment. The t time units period of system activity and fault tree observation is followed by a period of fault tree activity. This period lasts 0 time units, i.e. it happens in nonexistent time. Therefore, the modelled system does not do anything during this period, since it does not exist in reality but is only considered during the verification by model checking.

By using this time slicing technique, it becomes possible to model AND-nodes of the fault tree, because in this way the interval of time during which the hazards exist can be expressed. Furthermore, this approach makes it possible to reinitialize the state of the fault tree in a deterministic way. Usually, systems are prone to cause certain faults only for a certain period of time. These faults either occur within this certain period or they *never* occur. As the system considered here runs in a loop, it is adequate to define the time slice for the system as one turnaround of the loop. In general, it has to be determined for each leaf how long this particular threat lasts, but in the example presented in this paper the global time slicing approach turned out to be sufficient.

In the CSP implementations the time slicing is modelled by the timing events *begin* and *end* as follows: The time between a *begin* and an *end* is the time of system activity and fault tree observation. The time between an *end* and the following *begin* is used by the fault tree implementation to propagate the faults up the tree. Remember, this time period is nonexistent in reality.

In Fig. 4 a template of a leaf implementation in CSP is depicted. The process *LEAF_N* represents the n th leaf in the fault tree and waits for the timing event *begin* to occur. This timing event is triggered by the system when it starts a new loop. Together with the *end* timing event and the artificial events triggered by the environment, it builds the synchronized interface between the fault tree and the environment. After engaging in the *begin* event, the leaf accepts either an *end* event (no fault during this loop), or the artificial event *art_event_N* that triggers leaf n (a fault happened). In case of a fault, the process *CONSUME_art_event_{Ns}* “consumes” further occurrences of the artificial event until an *end* signals the end of system activity and beginning of fault tree activity. This is very important because the environment must not be influenced, especially it must not be blocked. Then, the leaf triggers the *signal_N* event which causes action in a node in an upper level in the fault tree. If no error occurred, the leaf is reset to its initial state. An example instantiation of a leaf will be given in Fig. 10 on page 14.

$$\begin{aligned}
LEAF_N &= begin \rightarrow (art_event_N \rightarrow CONSUME_art_event_Ns \\
&\quad \boxed{} \\
&\quad end \rightarrow LEAF_N) \\
CONSUME_art_event_Ns &= end \rightarrow signal_N \rightarrow LEAF_N \\
&\quad \boxed{} \\
&\quad art_event_N \rightarrow CONSUME_art_event_Ns
\end{aligned}$$

Fig. 4. Template of a fault tree leaf.

As mentioned above, the upper levels of the fault tree consist of OR-nodes or AND-nodes. The template implementation of an OR-node is depicted in Fig. 5. After the initial *begin* has introduced the first turnaround of the system loop, the process *OR_NODE_N* behaves as follows: It waits for the *end* timing event signalling the start of fault tree activity. If the following event is a *begin*, none of the lower nodes or leaves in the fault tree signalled an error and the process waits for the subsequent *end*. If *any* of the events gathered in the set *failset_NODE_N* happens, one of the lower nodes or leaves has signalled an error. Thus *OR_NODE_N* itself signals the event *signal_N* triggering a node in an upper level of the fault tree. Again, a “consuming” process is needed in order to cope with cases in which more than one lower node or leaf triggers *OR_NODE_N*.

$$\begin{aligned}
OR_NODE_N_INIT &= begin \rightarrow OR_NODE_N \\
OR_NODE_N &= end \rightarrow \\
&\quad (begin \rightarrow OR_NODE_N \\
&\quad \boxed{} \\
&\quad \quad \boxed{} \\
&\quad \quad \quad i:failset_NODE_N \rightarrow signal_N \rightarrow CONS_FAILS_NODE_N) \\
CONS_FAILS_NODE_N &= begin \rightarrow OR_NODE_N \\
&\quad \boxed{} \\
&\quad \quad \boxed{} \\
&\quad \quad \quad i:failset_NODE_N \rightarrow CONS_FAILS_NODE_N
\end{aligned}$$

Fig. 5. Template of a fault tree OR-node.

The modelling of the AND-nodes (cf. Fig. 6) of the fault tree is slightly more difficult than the one of an OR-node, as only the occurrence of *all* stimulating events of the node (represented as set *failset_NODE_N*) causes it to trigger the corresponding node on the higher level. Therefore, during the time period in which the system is active, the AND-node process *AND()* must accept all stimulating events in arbitrary order. If and only if each event of the initial failset *failset_NODE_N* already occurred (i.e. the remaining current failset *failset* is

empty), the upper level of the fault tree is triggered by the $signal_N$ event. After that, any surplus events are “consumed” by the $CONS_FAILS_NODE_N$ process until the timing event $begin$ sets off a new period of system activity and fault tree observation.

If the current failset is not yet empty, $AND()$ accepts any event of the initial failset. Afterwards, it behaves as if that special event had already occurred (cf. $AND(failset - \{i\})$ in Fig. 6).

$$\begin{aligned}
AND_NODE_N_INIT &= begin \rightarrow AND_NODE_N \\
AND_NODE_N &= end \rightarrow AND(failset_NODE_N) \\
AND(failset) &= \\
&\quad (card(failset) = 0) \ \& \ (signal_N \rightarrow CONS_FAILS_NODE_N) \\
&\quad (card(failset) > 0) \ \& \ (begin \rightarrow AND_NODE_N \\
&\quad \quad \quad \square \\
&\quad \quad \quad \square_{i:failset_NODE_N} (i \rightarrow AND(failset - \{i\}))) \\
CONS_FAILS_NODE_N &= begin \rightarrow AND_NODE_N \\
&\quad \quad \quad \square \\
&\quad \quad \quad \square_{i:failset_NODE_N} (i \rightarrow CONS_FAILS_NODE_N)
\end{aligned}$$

Fig. 6. Template of a fault tree AND-node.

Note that this technique of modelling and checking safety properties is more powerful than simply deriving a linear temporal logic (LTL) statement from the *structure* of the fault tree. The latter may only be used to specify static invariants which have to be always fulfilled by the system while our approach can deal with interval based real-time properties as well. When a leaf of the fault tree is triggered it is activated for the *period of one time slice*. If a different AND-connected leaf at the same hierarchy level is activated later but during the same time slice, the higher node will be triggered even if the first hazard does not exist any longer. In this way it is possible to model and check higher-level hazards which are based not only on the logical relations of hazards but also on their time relations. Therefore, the well-known algorithms to verify LTL statements (cf. for example [4]) are unsuitable to check that the highest node of the fault tree can never be triggered, since the fault tree cannot be expressed by an LTL formula.

4.3 Application: A Safety Layer for an Autonomous Wheelchair

The whole safety layer of the Bremen Autonomous Wheelchair has been developed using the approach described in this paper (cf. [12]). The complete fault tree has about 170 nodes and leaves, the highest levels are shown in Fig. 7. An

interesting aspect is the decomposition of the hazard “Collision” (1.1) which results in the distinction between active and passive collisions. The wheelchair itself is only able to avoid active collisions, i.e. collisions caused by the movements of the wheelchair. Passive collisions, e.g. caused by automatic doors which hit the wheelchair, have to be avoided by external safety mechanisms of the environment (indicated by the external hazard symbol). Therefore, demanding to avert this external hazard is one element of the set of requirements on the environment in which the wheelchair will operate safely. In this case, automatic doors are assumed to be “intelligent” enough to open and close only if this cannot cause a collision. The necessary safety mechanisms can also be designed with the approach described in this paper. The additional fault tree for automatic doors can be considered as part of the refinement of the “passive collision” external hazard.

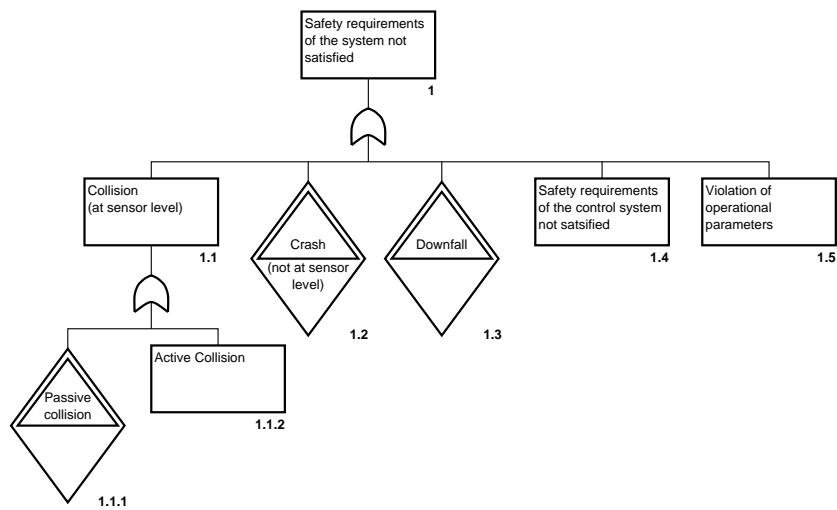


Fig. 7. The highest levels of the fault tree of the Bremen Autonomous Wheelchair.

The high levels of the fault tree deal with common hazards that are easy to adapt to other mobile robotics applications. As a consequence, the majority of the verification results might be re-usable as well.

As depicted in Fig. 8 the safety layer developed consist of a sensor/actuator module, net modules and the sensors and actuators of the wheelchair. Since many hardware devices used in robotics can only be used in combination with standard operating systems, it has to take into account that not all components can meet real time restrictions.

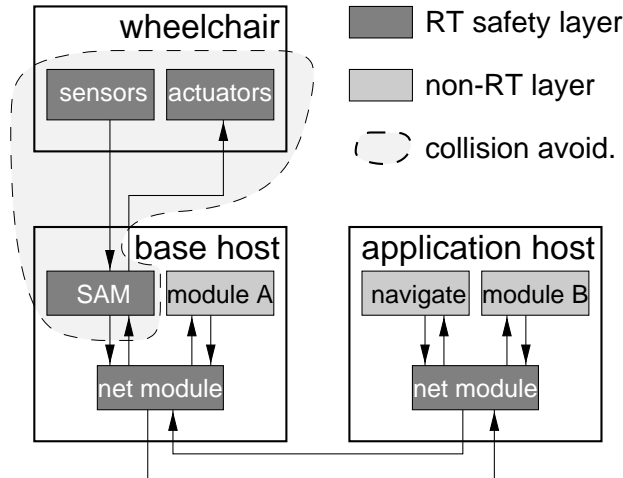


Fig. 8. The system architecture: the sensor/actuator module (SAM) is the interface to the real wheelchair. The net modules realize a real time communication between the hosts.

In the context of rehabilitation robotics, not only collisions with obstacles have to be taken into account as a catastrophic system state but also the *non-availability* of the system. For example, for a handicapped person it is essential that an autonomous wheelchair performs the expected task. In case of an emergency, a standstill cannot be considered as a safe system state since the user might have to get to an exit or a telephone. Therefore, reliable communication between basic system components becomes a safety-critical part. In contrast to the collision hazard, the specification of the communication protocol is confined to discrete-value/real-time properties.

Since the safety mechanisms to avert the “active collision” hazard are based on hybrid components and we are not yet able to handle the verification of such hybrid systems, the communication protocol will be analysed in detail in the sequel.

Requirements of the Protocol. In order to guarantee upper time bounds when using a multi-computer system it is necessary to use a hard real time communication protocol. On the other hand, it is preferable to use cheap standard technology instead of specialized hardware. Therefore, the protocol is realized as an additional layer on top of an ethernet network.

Because ethernet is based on a CSMA/CD protocol the new layer has to avoid collisions on the network. This is realized by a time division multiple access (TDMA) frame-protocol (cf. [29]) which ensures that at any time only one computer can write to the bus. Besides, it is important that every host *period-*

ically has the opportunity to send data. The resulting fault tree segment and its CSP specification is presented in the next section, followed by the model of the environment and a naive realization of the producer and consumer processes (the EUC).

A Fault Tree Segment in CSP. The CSP specification of the fault tree segment for communication problems (see Fig. 9 for an extract) can be realized by using the templates for leaves and nodes shown in section 4.2. The node X represents an OR-node which has to agree on fault-triggers from the nodes $X.1$ to $X.3$, i.e. the set $failset_NODE_X$ contains the events $signal_X.1$ to $signal_X.3$. Accordingly, the node $X.1$ has to be modelled as an AND-node reacting to $signal_X.1.1$ and $signal_X.1.2$.

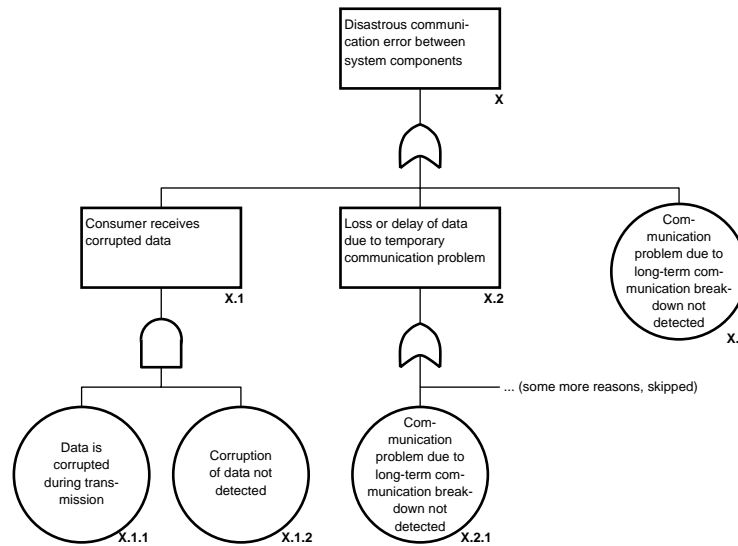


Fig. 9. Extract of a fault tree segment dealing with communication problems.

In order to describe the instantiation of a leaf we focus on the basic hazard $X.2.1$ — the delay of data might be caused by a collision on the network. Therefore, the model of the network has to provide an artificial event that signals a collision (see section 4.3). The resulting leaf is shown in Fig. 10.

The fault tree process $FAULTTREE$ is composed of node and leaf processes running in parallel. All of them have to agree on the time slicing events *begin* and *end*, the different levels are synchronized by the internal fault tree signals as well. Finally, the whole process has to be synchronized with the system process and the environment by the artificial events to allow the observation of the system

$$\begin{aligned}
LEAF_X_2_1 &= begin \rightarrow (art_event_Coll \rightarrow CONSUME_art_event_Cs \\
&\quad \square \\
&\quad end \rightarrow LEAF_X_2_1) \\
\\
CONSUME_art_event_Cs &= end \rightarrow signal_X_2_1 \rightarrow LEAF_X_2_1 \\
&\quad \square \\
&\quad art_event_Coll \rightarrow CONSUME_art_event_Cs
\end{aligned}$$

Fig. 10. Instance of a fault tree leaf in CSP for detecting collisions of data packets on the network.

behaviour. In order to verify that the system is correct, it is sufficient to show that the highest fault tree node can never trigger the corresponding event (here: $signal_X$, see Fig. 11). This is done by a refinement check in the trace model against the process *GOODSYS*.

Modelling of the Environment. The specification of the ethernet process shown in Fig. 12 accepts data on the net_send channel and passes it to the net_recv channel. The $end_transmit$ and end_recv events model the time it takes to send and receive messages, respectively. Whenever more than one net_send event occurs before all participating processes agree on the $end_transmit$ event, a collision occurs. This is signalled to the fault tree process by triggering the artificial event art_event_Coll . After a collision has occurred all subsequent net_send events are ignored.

A simple producer process (*PROD* in Fig. 13) nondeterministically decides whether to put some data on the network or not. It ignores all activities during the receive phase. Correspondingly, a consumer process always accepts transmissions after the transmit phase has ended.

Drawbacks of a Simple Protocol. It should be pointed out that the environment introduced above is prone to behave maliciously, i.e. it may trigger one of the artificial events that cause the propagation of faults in the fault-tree process *FAULTTREE* and finally generates the $signal_X$ event.

An environment consisting of two producers *PROD*(1) and *PROD*(2), a consumer *CONS* and the underlying network *NET* behaves faulty if the two producers simultaneously try to transmit data and thus cause a collision of packets on the network. As expected, a trace refinement proves that the sequence of events depicted in Fig. 14 is possible and delivers the highest fault-signal of the fault-tree ($signal_X$).

The period of system activity starts with the timing event $begin$ (cf. 14). Then, each producer generates data and transmits it by engaging in the particular $net_send.x$ event. As shown, the second producer transmits its data before

$$\begin{aligned}
FAULTTREE &= (((LEAF_X_1_1 \\
&\quad \parallel LEAF_X_1_2) \\
&\quad \{begin,end\} \\
&\quad \parallel LEAF_X_2_1) \\
&\quad \{begin,end\} \\
&\quad \parallel \dots) \\
&\quad \{begin,end\} \\
&\quad \parallel \\
&\quad \{signal_X_1,signal_X_2,signal_X_2_1,\dots,begin,end\} \\
&\quad ((AND_NODE_X_1_INIT \\
&\quad \parallel OR_NODE_X_2_INIT) \\
&\quad \{begin,end\} \\
&\quad \parallel OR_NODE_X_3_INIT)) \\
&\quad \{begin,end\} \\
&\quad \parallel \\
&\quad \{signal_X_1,signal_X_2,signal_X_3,begin,end\} \\
&\quad OR_NODE_X_INIT \\
FTSYS &= ESYS \quad \parallel \quad FAULTTREE \\
&\quad \{art_event_Coll,art_event_...,begin,end\} \\
GOODSYS &= CHAOS(\Sigma - \{signal_X\})
\end{aligned}$$

Fig. 11. Construction of the fault tree system *FTSYS*. It can be used to verify that the process *ESYS* (consisting of the system and its environment) cannot perform traces which lead to the top hazard in the fault tree, i.e. to ensure that the OR-node *X* never triggers the event *signal_X*.

$$\begin{aligned}
NET &= begin \rightarrow (net_send?data \rightarrow (end_transmit \\
&\quad \rightarrow net_recv!data \\
&\quad \rightarrow end_recv \\
&\quad \rightarrow end \rightarrow NET \\
&\quad \square \\
&\quad net_send?data \\
&\quad \rightarrow art_event_Coll \\
&\quad \rightarrow CONSUME_SENDS; end_recv \\
&\quad \rightarrow end \rightarrow NET) \\
&\quad \square \\
&\quad end_transmit \rightarrow end_recv \rightarrow end \rightarrow NET) \\
CONSUME_SENDS &= net_send?data \rightarrow CONSUME_SEND \\
&\quad \square \\
&\quad end_transmit \rightarrow SKIP
\end{aligned}$$

Fig. 12. The abstract specification of the underlying ethernet network.

$$\begin{aligned}
PROD(id) &= (produce_Item.id \rightarrow net_send!id \rightarrow end_transmit \\
&\quad \rightarrow end_recv \rightarrow PROD(id)) \\
&\quad \sqcap \\
&\quad (end_transmit \rightarrow end_recv \rightarrow PROD(id)) \\
\\
CONS &= end_transmit \rightarrow (net_recv?id_sent \rightarrow end_recv \\
&\quad \rightarrow consume_Item.id_sent \rightarrow CONS) \\
&\quad \sqcap \\
&\quad end_recv \rightarrow CONS)
\end{aligned}$$

Fig. 13. Specification of producer and consumer processes.

*begin, produce_Item.2, produce_Item.1, net_send.1, net_send.2,
art_event_Coll, end_transmit, end_recv, end,
signal_X.2.1, signal_X.2, signal_X*

Fig. 14. Trace leading to the top hazard, indicated by event *signal_X*

the data of producer 1 have been received. The resulting collision is signalled by the *NET* process when issuing the artificial event *art_event_Coll*. After the period of system activity ended (indicated by the *end* event), the leaf X.2.1 triggers its fault-event. This is propagated up the fault tree until the occurrence of the top hazard *signal_X* is signalled.

In order to deal with this problem, the system has to be extended by safety mechanisms, as described in the sequel.

Frame-Protocol with Smart Buffering. As indicated, collisions on the network can be avoided by employing a time division approach, i.e. by strictly assigning the permission to transmit data to only one host at a time. Thus, each attempt to send data will succeed immediately and will not be delayed by the CSMA/CD retransmission technique that is used in case of packet collisions. The frame-protocol is controlled by a dedicated computer in the network, the *master*. Periodically, it requests data of a specific ID by broadcasting. The computer hosting the application that produces such data answers the request by broadcasting the corresponding data to all hosts in the net. This procedure works as long as the data IDs are unique, i.e. it must never happen that two hosts feel responsible to answer the same request.

The applications on the hosts should not be allowed to access the net directly. As, furthermore, it should not be taken for granted that these high level modules meet hard real time restrictions, a double buffering policy is employed. Data received by the net module implemented within the net interrupt is copied into a *net buffer*. During the so-called *housekeeping period* no net activity is allowed and the data from the net buffer is copied to an *application buffer* which is

a piece of memory shared between all applications and the net module. Thus, upper time bounds can be defined specifying how long it takes for the data sent by an application to be received by another application on an arbitrary remote host.

To avoid race conditions with regard to access to the shared memory area between the net module and the applications, a procedure is needed to ensure mutual exclusion. As the net module has to access the memory without delay (hard real time conditions), simple locking algorithms fail. This problem is an example for the necessity of iterative fault tree development, since the chosen safety mechanism (buffering with simple locking techniques) causes a new threat (memory access blocked) that has to be taken into consideration in the fault tree. By introducing an artificial event that is triggered by the safety mechanism whenever two applications try to access the same buffer at the same time it can be shown, by model checking, that an extended system including such simple locked buffers fails.

In [12] a smart buffering concept has been developed that allows the net module and all applications to access the shared memory area without delay. The basic idea is to provide $\#readers + \#writers$ times as much memory as usually needed for a buffer, i.e. each reading application accesses its own part of the memory while the net module is able to write into another memory area. It has been verified that this approach ensures mutual exclusion and guarantees the delivery of data within bounded time even if the applications do not meet the real time requirements.

Eventually, it has been shown that the top hazard of the fault tree depicted in Fig. 9 never occurs when employing the frame-protocol in combination with the smart buffering technique.

4.4 Conclusive Remarks

An informal specification of hazards is employed in order to formalize the undesired behaviour of a system and its environment in a hierachically structured fault tree. The fault tree specification is the only reference document required to verify safety properties. It is shown that the chosen approach is an adequate means to prove that a designed safety system can avert all relevant hazards. As segments of the fault tree can be re-used for similar application domains this method facilitates the re-use of verification results.

We are convinced that the integration of informal requirements, the semi-formal fault tree technique and the formal verification using a model checker is one of the strong points of this method. The stepwise increase of the level of formalization makes it possible to describe the requirements in an appropriate way at each stage of development. During the fault tree based hazard analysis it might not be clear what certain informal descriptions exactly mean (*How* close is “close”?). Nevertheless, the formal structure of the fault tree allows a thorough reasoning about the completeness of the specification. When formalizing the hazards of the fault tree to safety requirements and later deriving safety mechanisms, problems may occur resulting from ambiguous specifications. The

resulting subsequent reiteration of the fault tree analysis often yields important new design criteria, as shown in section 4.3.

5 Conclusion and Future Work

Providing a specification of the environment is a major property demanded from a formal method that is used to develop a safety controller for a robotic application. The presented approach is well-suited to systematically derive environment requirements in combination with safety mechanisms for the developed system. Furthermore, the verification method described here is capable of handling interval-based real-time specifications.

Nevertheless, it does not yet suit arbitrary problem domains, e.g. if hybrid parts are involved. Thus, two extensions are proposed in order to further broaden the applicability of the approach in the future.

5.1 Improved Fault Tree Semantics

The additional incorporation of a fault tree that not only uses a formal structure but also has a formal semantics realized by formally specified leaves, would ease the following deduction of safety requirements significantly (it should be possible to automate this step). However, the informal description of hazards in the first fault tree document should be retained, because it is an excellent base for the communication between the software designer and the customer.

Often, it is desirable to express time dependencies of hazards in the fault tree in a formal way, i.e. an extension of the fault tree syntax and semantics used here that makes it possible to specify time relations between stimulating branches of AND-nodes. Such a semantics would allow to enhance the implementation of the fault tree by introducing a means to assign detailed time periods to specific nodes instead of using a global time slicing in the form of *begin* and *end* events. A similar approach is presented in [5], where conventional fault trees are formalised and analysed with the minimal cut set approach and with timed petri nets. We will examine if this method can be incorporated into our CSP based verification.

Moreover, facilities for expressing and validating/verifying/testing hybrid aspects of robotics applications are needed. Since the presented verification approach is based on CSP, a compatible hybrid extension should be realized either in Hybrid-CSP (cf. [31]) or with hybrid automata, which can be combined with the CSP world (cf. [1]).

5.2 Robotics and Test Derivation

Many “real-world” applications turn out to be too large to make verification of safety properties possible. Here it is necessary to gain confidence in the system by performing tests. The test cases have to be chosen in a systematic way to ensure that all relevant parts of the system are tested. Therefore, it is desirable to use a test tool, such as RT-Tester (formerly called VVT-RT), that is based on formal

methods and selects the test cases so that the test coverage gradually converges to completeness (cf. [17,18]). One example of such a “real-world” application is the safety layer for the autonomous wheelchair described above. Similar to the verification approach presented here, the fault tree can be used to evaluate its behaviour. Furthermore, the fault tree specifies the acceptable deviations from the normal behaviour of the environment, e.g. the number of consecutive sensor misreadings the system has to cope with. This information can be exploited by the test tool to generate all relevant test cases. Moreover, RT-Tester can already handle the automated test of general hard real-time properties specified in Timed-CSP and it can be used to check the behaviour of hybrid systems (cf. [22]). Thus, the test tool RT-Tester supports all means necessary for the formal *testing* of robotics applications.

References

1. Amthor, P.: A CSP Model for Hybrid Automata. Proceedings of the 3. BCS-FACS Northern Formal Methods Workshop, Ilkley, UK (1998)
2. Bell, D. A., Levine, S. P., Koren, Y., Jaros, L. A., Borenstein, J.: Design Criteria for Obstacle Avoidance in a Shared Control System. Proc. RESNA '94 Conference. Nashville (1994)
3. Formal Systems Ltd.: Failures-Divergence Refinement. FDR2 User Manual, Oxford (1997)
4. Gerth, R. et al.: Simple On-the-fly Automatic Verification of Linear Temporal Logic. Proc. of the Fifteenth International Symposium on Protocol Specification, Testing and Verification, Warsaw (1995)
5. Gorski, J., Wardzinski, A.: Timing Aspects of Fault Tree Analysis of Safety Critical Systems. Proc. of the 5th Safety-critical Systems Symposium. Brighton, England (1997).
6. Hamilton, D. L., Visinsky, M. L., Bennett, J. K., Cavallaro, J. R., Walker, I. D.: Fault-Tolerant Algorithms and Architectures for Robotics. Proc. 1994 IEEE Mediterranean Electrotechnical Conference. Antalya, Turkey (1994).
7. Hansen, K. M.: Linking Safety Analysis to Safety Requirements. PhD Thesis, DTU Technical University of Denmark (1996)
8. Hoare, C. A. R.: Communicating Sequential Processes. International Series in Computer Science, Prentice Hall (1985)
9. International Electrotechnical Commission: International Standard IEC 1025 Fault Tree Analysis (FTA). Geneva (1990)
10. Kazerooni, H.: Human Power Extender: An Example of HumanMachine Interaction via the Transfer of Power and Information Signals. Proc. AMC 98 - 5th Int. Workshop on Advanced Motion Control, Coimbra University Portugal (1998).
11. Kotonya, G., Sommerville, I.: Integrating Safety Analysis and Requirements Engineering. Proc. Joint Asia Pacific Software Engineering Conference (APSEC) and International Computer Science Conference (ICSC) 1997. Hongkong
12. Lankenau, A., Meyer, O.: Der autonome Rollstuhl als sicheres eingebettetes System. Masters Thesis, Universität Bremen (1997). <http://www.informatik.uni-bremen.de/~emm/DA/DA-Abstract.html>
13. Lankenau, A., Meyer, O., Krieg-Brückner, B.: Safety in Robotics: the Bremen Autonomous Wheelchair. Proc. 5th International Workshop on Advanced Motion Control, Coimbra (1998)

14. Laprie, J.C.: Dependability, Basic Concepts and Terminology. Dependable Computing and Fault-Tolerant Systems Vol.5, Springer (1992)
15. Leuschen, M.: Robot Reliability Through Fuzzy Markov Models (MSc Thesis). Rice University, Texas, USA (1997).
16. Leveson, N.: Safeware: System Safety and Computers. Addison-Wesley, Reading, MA (1995)
17. Peleska, J.: Formal Methods and the Development of Dependable Systems. Habilitationsschrift, Christian-Albrechts-Universität zu Kiel (1996). <http://www.informatik.uni-bremen.de/~jp/papers/habil.ps.gz>
18. Peleska, J.: Testing Reactive Real-Time Systems. Tutorial, held at the FTRTFT 98. Denmark Technical University, Lyngby (1998).
19. Pires, G., Arajo, R., Nunes, U., de Almeida, A.T.: ROBCHAIR – A Powered Wheelchair Using a Behaviour-Based Navigation. Proc. AMC 98 – 5th Int. Workshop on Advanced Motion Control, Coimbra University Portugal (1998).
20. Röfer, T., Lankenau, A.: Architecture and Applications of the Bremen Autonomous Wheelchair. Fourth Joint Conference on Information Systems. Research Triangle Park, North Carolina, USA (1998)
21. Roscoe, A. W.: The Theory and Practice of Concurrency. International Series in Computer Science, Prentice Hall (1998)
22. Schlingloff, H., Meyer, O., Hülsing, Th.: Correctness Analysis of an Embedded Controller. Proceedings of the 10th DASIA Conference. Lisbon, Portugal (1999) (to appear)
23. Seward, D. W., Bradley, A., Margrave, F. W.: Hazard Analysis Techniques for Mobile Construction Robots. Proc. 11th Int. Symp. on Robotics in Construction. Brighton, England (1994).
24. Seward, D. W., Margrave, F. W., Sommerville, I., Kotonya, G.: Safe Systems for Mobile Robots – The Safe-SAM project. Achievement and Assurance of Safety. Proc. Third Safety-critical Systems Symposium. Brighton, England (1995).
25. Seward, D. W., Margrave, F. W., Sommerville, I., Morrey, R.: LUCIE the Robot Excavator – Design for System Safety. Proc. IEEE Int. Conference on Robotics and Automation. Minneapolis, USA (1996).
26. Sommerville, I., Seward, D., Morrey, R., Quayle, S.: Safe Systems Architectures for Autonomous Robots. Proc. Third Safety-critical Systems Symposium, Brighton, England (1995).
27. Storey, N.: Safety-Critical Computer Systems. AddisonWesley (1996).
28. Vesely, W. E. et al.: Fault Tree Handbook. Nuclear Regulatory Commission, Office of Nuclear Regulatory Research, Division of Systems and Reliability Research (1981). ISBN 0-16-005582-2.
29. Tanenbaum, A. S.: Modern Operating Systems. Prentice Hall (1992)
30. Zhang, Y.: A Foundation for the Design and Analysis of Robotic Systems and Behaviors (PhD Thesis). University of British Columbia, Kanada (1994).
31. Chaochen, Z., Ravn, A.P., Hansen, M.R.: An extended Duration Calculus for Hybrid Real-time Systems. In: R.L. Grossman, A. Nerode, A.P. Ravn and H. Rischel (eds.): Hybrid Systems. Springer LNCS 763 (1993)

Formal Methods in Robotics: Fault Tree Based Verification

Oliver Meyer
Axel Lankenau

University of Bremen, Germany
Bremen Institute of Safe Systems

QualityWeek Europe '99



Overview

Formal vs. Informal Methods (in Robotics)

Fault Tree Analysis

The Bremen Autonomous Wheelchair

Verification of Safety Requirements

Example

Conclusion

QualityWeek Europe '99



Definitions

(Informal) Validation

Check that *First* Design Document is Complete and Consistent with Respect to Intended Purpose

(Formal) Verification

Mathematical Proof that Requirements are Preserved During Design Steps

Test

Spot Check Analysis of the Behaviour of the Real System, Steadily Increasing the Confidence

QualityWeek Europe '99



Formal Methods

Precise Description of Requirements

- Facilitates Unambiguous Contracts
- Allows Mathematical Treatment
- Indispensable for Verification
- Tool Support & Automation of Development

State of the Art for Safety-Critical Systems

Increasing Success in Industry

QualityWeek Europe '99



Informal Methods

Complement the Formal Description

- Ease the Discussion with Customers
- Sufficient for Non-Safety-Related Aspects

Computers cannot Grasp their Meaning



Combination of Both Worlds

QualityWeek Europe '99



Formal Methods in Robotics

Ensure: No Catastrophic Behaviour



Rehabilitation Robotics

- No Collisions
- User Commands must be Executed if Possible
- Classical Safety + Availability + Reliability

QualityWeek Europe '99

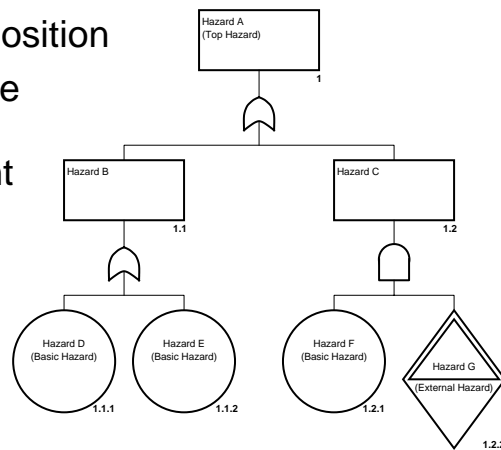


Fault Tree Analysis

Structural Decomposition
of the Threat Space

Hazard Refinement

Fault Propagation



QualityWeek Europe '99



Fault Trees in Robotics

Specification of Undesired Behaviour

- Requirements Imposed on Controller
- Requirements Imposed on Environment

Re-Use of High-Level Tree Fragments

- Similar Hazards in other Robotics Domains
- Generic Frameworks Inferable

QualityWeek Europe '99



Bremen Wheelchair “Rolland”

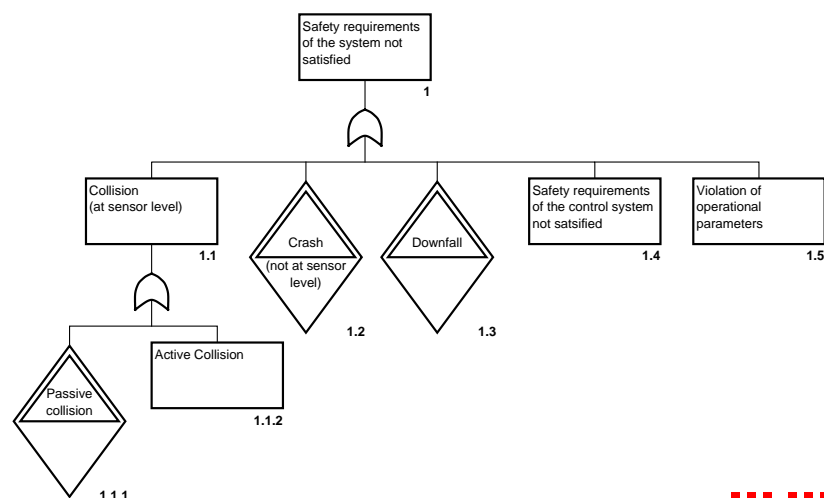


- Common Power-Wheelchair with Extra Sensors and PC
- Increases the Mobility of Handicapped People
- Safety Layer Averts Collisions
- Support for Difficult Manoeuvres (Door Passage)
- Learning of Routes in Office Buildings
- Shared-Control System

QualityWeek Europe '99



Fault Tree Example



QualityWeek Europe '99

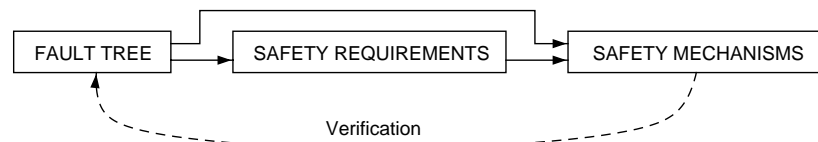


Fault Trees and Safety

Conventional Use of Fault Trees

- Derive Safety Requirements
- "Guess" Adequate Safety Mechanisms

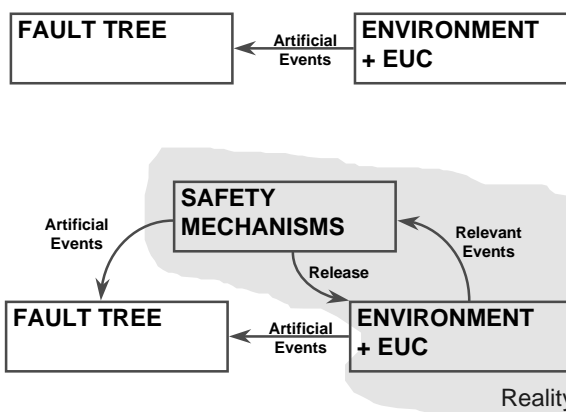
Verification against Requirements needed



QualityWeek Europe '99



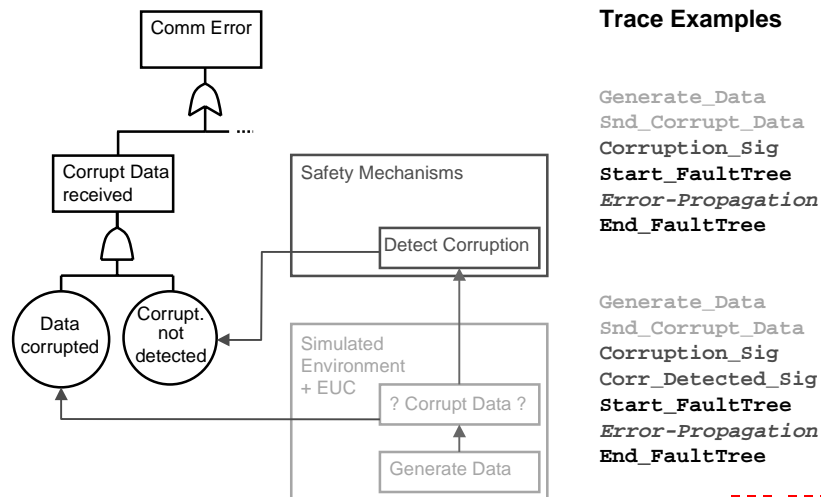
Verification Approach



QualityWeek Europe '99



Animated Example



Trace Examples

```
Generate_Data  
Snd_Corrupt_Data  
Corruption_Sig  
Start_FaultTree  
Error-Propagation  
End_FaultTree
```

```
Generate_Data  
Snd_Corrupt_Data  
Corruption_Sig  
Corr_Detected_Sig  
Start_FaultTree  
Error-Propagation  
End_FaultTree
```

QualityWeek Europe '99



Model Checking in CSP

Automated Verification Method

- Exhaustive Search of the State Space
- Allows to Check Freedom of Deadlocks and Livelocks, Reachability and Refinements
- "Press Button" Approach

Communicating Sequential Processes

- Event Based Specification Language
- Model Checking Tool FDR

QualityWeek Europe '99



Fault Tree Leaf in CSP

Leaves and Nodes Modelled as Processes

Leaves Store Signals for one Cycle

Leaf Example in CSP:

```
LEAF = Start_FaultTree -> (Corruption_Sig -> CONSUME_CORR
    []
    End_FaultTree -> LEAF)
```

```
CONSUME_CORR = End_FaultTree -> Error_Prop_Sig -> LEAF
    []
    Corruption_Sig -> CONSUME_CORR
```

QualityWeek Europe '99



Verification Approach in CSP

- Fault Tree is Implemented by a Collection of Leaves and Nodes Running in Parallel
- System and Fault Tree are Executed in Parallel
- Proof Obligation is Simplified to
“Top Hazard does not Occur”

QualityWeek Europe '99



Conclusion

Presented Approach Supports

- Stepwise Formalisation of Requirements
- Separation of Controller and Environment Specifications
- Reasoning about Completeness

“Press Button” Verification

Case Study in Rehabilitation Robotics

QualityWeek Europe '99



Future Work

Complex Timing Conditions

- Formalisation of Timing Conditions
- Temporal Relations between FT Branches

Hybrid Specifications

Fault Tree Based Testing (RT-Tester)

- Environment Spec. for Test Generation
- Controller Spec. for Test Evaluation

Probabilistic Fault Trees

QualityWeek Europe '99



Test Automation in Telecommunications Software: a Case Study on GPRS

R. DELMIGLIO⁽¹⁾, A. MANINI⁽¹⁾, G. BAZZANA⁽²⁾, G. RUMI⁽²⁾, F. BASILI⁽²⁾, E. BENDINELLI⁽³⁾, A. RAPPELLI⁽³⁾

(1) Italtel Spa

Reti Mobili Research and Development (RM-RD) department
SS11, Km 158 – 20060 Cassina De Pecchi (Milan) – Italy
Web: <http://www.italtel.it>
Phone: +39-02-4388-6162
Fax: +39-02- 4388-6872

(2) ONION Spa

Communications, Technologies, Consulting
Via L. Gussalli, 9 – 25131 Brescia – Italy
Web: <http://net.onion.it>
Phone: +39-030-3581510
Fax: +39-030-3581525

(3) Spazio ZeroUno Spa

Via Grande, 21
20090 Vimodrone (Milan - Italy)
Web: <http://www.spaziozerouno.it>
Phone: +39-02-2650701
Fax: +39-02-2650740

ABSTRACT

This paper describes the test automation experiences performed at Italtel Reti Mobili – Research and Development (RM-RD) department focusing on the adoption of specific methods and tools for GPRS service.

The paper is organised around the following topics:

- a short company profile and a global description of the evolution of mobile telephony;
- the reference development process, the testing practices and the Software Process Improvement (SPI) activities within which test automation activities have been experienced;
- an overview about the GPRS service in terms of technical foundations and envisaged business benefits;
- the test automation activities, including a technical description of the environment and a quantitative effectiveness analysis of the benefits introduced by the adoption of the selected practices and tools;
- a detailed technical description of the methods and tools internally designed and developed by RM-RD department to drive the testing of the GPRS feature;
- a general appraisal of lessons learnt and future goals.

TABLE OF CONTENTS

| | |
|---|-----------|
| ABSTRACT | 1 |
| TABLE OF CONTENTS | 2 |
| 1 COMPANY PROFILE..... | 3 |
| 2 EVOLUTION OF GSM AND MOBILE TELEPHONY | 4 |
| 2.1 THE STANDARDS..... | 4 |
| 2.2 THE SYSTEMS..... | 5 |
| 3 THE REFERENCE SOFTWARE DEVELOPMENT AND TESTING PROCESSES | 6 |
| 3.1 SOFTWARE LIFE CYCLE..... | 6 |
| 3.2 TESTING ACTIVITIES..... | 8 |
| 4 TEST AUTOMATION IN THE CONTEXT OF SOFTWARE PROCESS IMPROVEMENT | 9 |
| 5 DATA TRANSMISSION ON MOBILE NETWORKS | 11 |
| 5.1 HSCSD AND GPRS..... | 11 |
| 5.2 GPRS System Architecture..... | 12 |
| 6 TEST AUTOMATION ENVIRONMENT FOR GPRS TESTING | 15 |
| 6.1 HATT (HOST AUTOMATION TEST TOOL) | 16 |
| 6.2 LSU Plus (LINE SERVER UNIT PLUS) | 17 |
| 6.3 GTAS (GSM TEST AUTOMATION SYSTEM)..... | 17 |
| 6.4 Abis Applications | 19 |
| 6.4.1 LabLog..... | 19 |
| 6.4.2 LabRes | 19 |
| 6.4.3 Lgts Plus (Line Server Unit GSM Based Traffic Simulator Plus)..... | 19 |
| 6.5 Gb Applications..... | 22 |
| 6.5.1 GbLog..... | 22 |
| 6.5.2 GbSeq | 22 |
| 6.5.3 GbTraffic | 22 |
| 7 PRELIMINARY RESULTS AND THE WAY AHEAD..... | 23 |
| ACKNOWLEDGEMENTS | 24 |
| REFERENCES | 24 |

1 COMPANY PROFILE

Italtel Spa, the largest Italian telecommunications manufacturer, is an European supplier of telecommunication networks and systems, operating on the global market.

Italtel designs, manufactures, markets and installs systems and equipment for public and private applications. In Italy and abroad it implements systems and networks on a turnkey basis. The company is active, full-line, in every field of telecommunications.

Research and Development is crucial to Italtel's competitiveness on the world markets; in the last year Italtel's commitment to R&D Activities has been intensive with investments representing 13% of sales revenues.

The Mobile Network Business Unit has the following mission:

- to be a turn key mobile network supplier;
- to develop the business of mobile networks;
- to be the competence centre for the whole Italtel-Siemens group for Base Station Subsystems used within cellular networks and wireless access systems;
- to support customers in all project implementation phases;

As of date April 1999, over 140.000 Italtel-Siemens TRX are operational world-wide in the countries highlighted by the following figure.

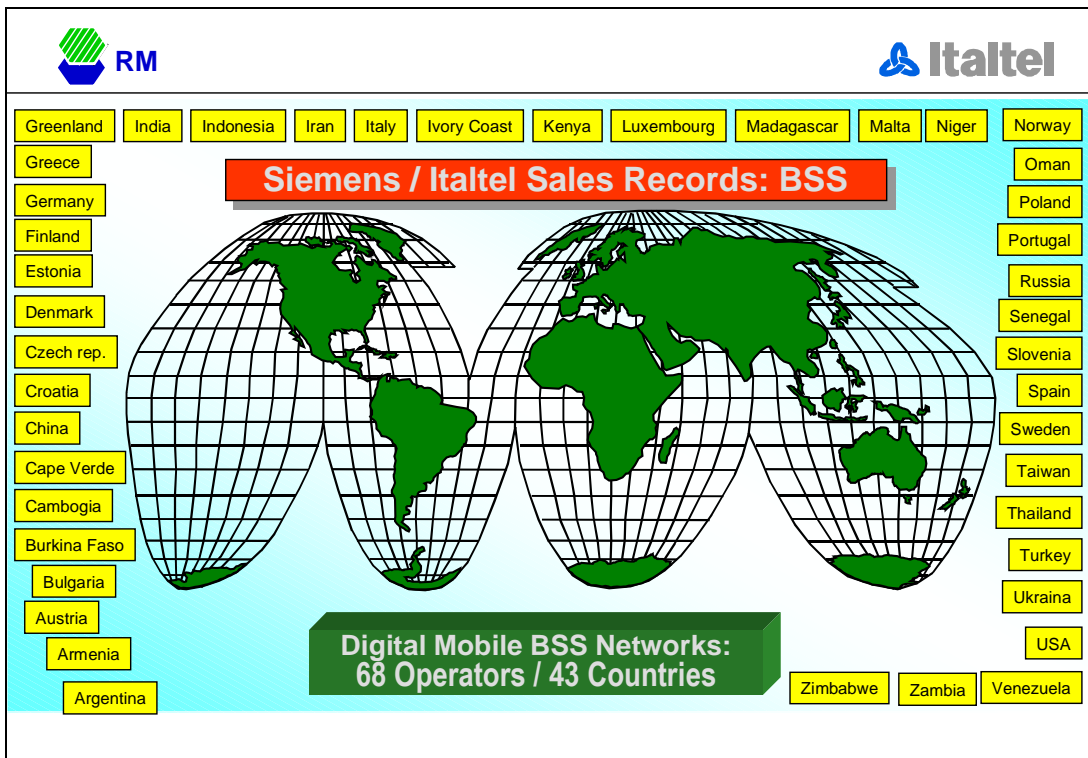


Figure 1.1 – Siemens/ Italtel BSS installations

For more details on the company, the interested reader is referred to the Italtel WWW at the URL <http://www.italtel.it/>

2 EVOLUTION OF GSM AND MOBILE TELEPHONY

2.1 THE STANDARDS

Originally, the GSM standard was intended as a comprehensive standard, to be used until the standardisation of a 3rd generation follow-up technology. Early in late 80's it became obvious that it was not possible to freeze all the technical details and to fully standardise service requirements. This resulted in the important decision to leave the GSM standard open and to develop and work on it permanently instead. The evolutionary GSM concept thus provides enough scope for technical evolutions and can be quickly adapted to the rapidly changing market conditions. GSM developed in various phases, briefly sketched in the following.

GSM Phase 1

Agreed upon in 1990/1991, it includes all basic prerequisites for mobile and digital transmission of information. Speech transfer plays an important role and data transmission was defined with rates from 0,3 to 9,6 kbit/sec. In this phase only a few supplementary services were defined such as: *call forwarding* (i.e. subscriber busy, un-reachable or does not answer) and *call barring* (i.e. all calls, international calls, incoming calls, etc.).

GSM Phase 2

Research on this phase was concluded in 1995. Supplementary services were specified, including *SMS* (short message service), *calling/ connected line identity presentation* (displays calling party's directory number before/ after call connection), *calling/connected line identity restriction* (restricts the display of the calling party's number at called party's side before/ after call connection), *call waiting* (informs the user about a second incoming call and allows to answer it), *call hold* (puts an active call on hold in order to answer or originate another call), *multiparty communication* (conference calls), *closed user group* (establishment of groups with limited access), *advice of charge*, *unstructured supplementary services data* (offers an open communications link for use between network and user for operator-defined services), *operator-determined barring* (restriction of different services, call types by the operator).

Of central importance was the agreement on downward compatibility, meaning that all networks and terminal equipment of phase 2 were compatible to the networks and terminal equipment of phase 1.

GSM Phase 2 plus

GSM phase 2 plus addresses evolved requirements for mobile radio systems: improved speech quality, granted by the introduction of a new speech code (*Enhanced Full Rate Speech*), and worldwide availability, achieved through multi-mode terminal equipment (*satellite roaming*).

Referring to the implementation of "*mobile computing*"/ *Internet access*, bearer services¹ such as High Speed Circuit Switched Data (*HSCSD*), and General Packet Radio Service (*GPRS*) are standardised allowing for the adaptation of transmission rates to those of ISDN. The importance of phase 2 plus lies also in the creation of a platform on which the GSM follow-up standard UMTS (Universal Mobile Telecommunication Standard) will be based.

UMTS (Universal Mobile Telecommunication Standard)

The 3rd mobile radio generation currently being standardised under the heading IMT-2000 (International Mobile Telecommunication) designates a global system of compatible standards

¹ The Bearer Services are the basis of the safe data transmission between the interfaces of the terminal equipment

which indeed is able to meet the high demands placed on future mobile radio systems. The general aim is to enable “communication with anyone, anywhere, anytime”.

In the framework of IMT-2000 guidelines ETSI is about to standardise a follow-up GSM standard based on the experiences with and the success of GSM: the standard is known as UMTS. UMTS is a downward-compatible GSM standard of the 3rd mobile radio generation; as such it shall provide worldwide multimedia access at any point in time and cover all current mobile radio applications. Data rates of 8 kbit/s up to a maximum of 2Mbit/s shall be supported.

2.2 THE SYSTEMS

From the architectural point of view, a GSM system is quite a complex object, since it has to deal with multi-services and with the peculiarities of cellular networks. Looking at the system from the outside, GSM is in direct contact with users, with other telecommunications networks and with the personnel of the service providers.

The internal GSM architecture distinguishes three parts (see fig 2.1): the **BSS** (Base Station Sub-System), that is in charge of providing and managing transmission paths, the **NSS** (Network and Switching SubSystem), that is in charge of managing the communications and the **OSS** (Operation SubSystem) which provides the interface to the system for the network operator.

Getting into details of the BSS, we can find the following Network Elements:

- a transmission equipment (**BTS** – Base Transceiver Station);
- a managing equipment (**BSC** – Base Station Controller);
- a speech encoding/decoding equipment (**TRAU** – Transcoder and Rate Adapter Unit)

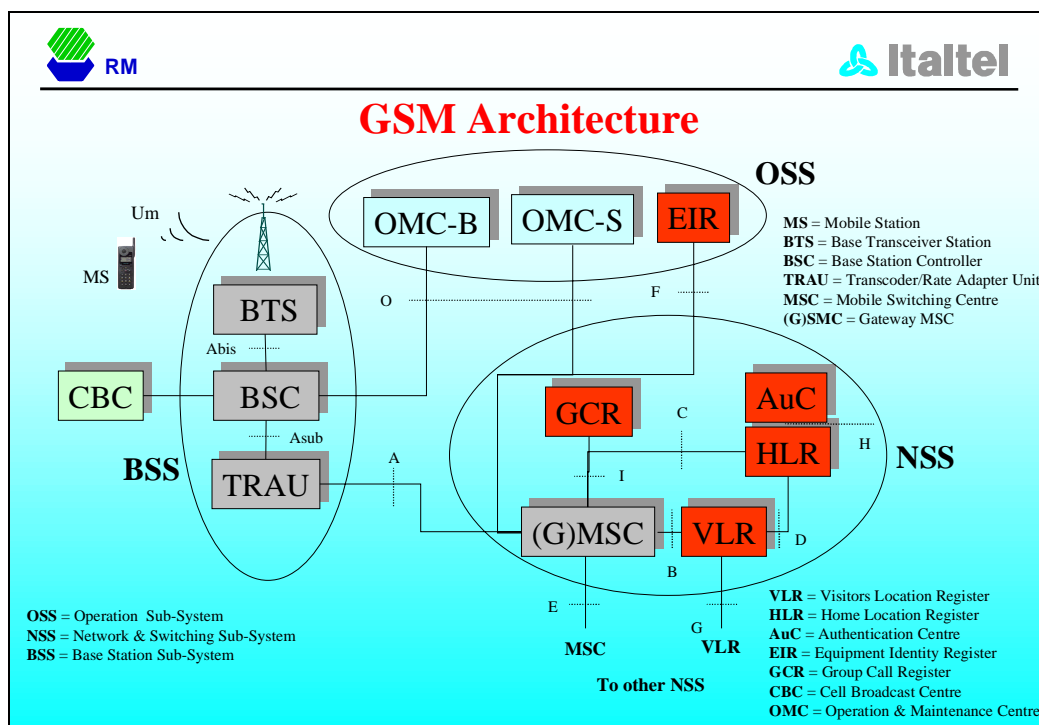


Fig 2.1 – GSM System architecture

The following figure shows the achieved and planned Italtel BSS line development to cope with the evolution of standards and services

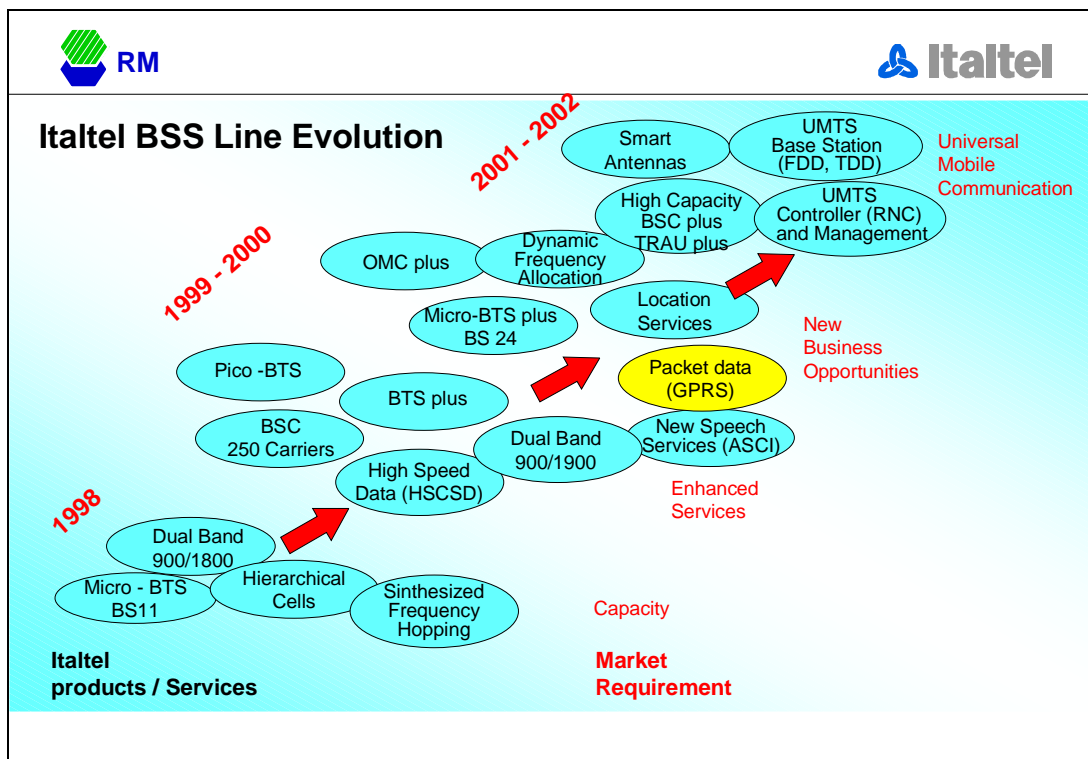


Fig 2.2 – Italtel BSS line evolution

3 THE REFERENCE SOFTWARE DEVELOPMENT AND TESTING PROCESSES

3.1 SOFTWARE LIFE CYCLE

The software development life cycle adopted at Italtel RM-RD can be summarized as follows (see figure 3.1):

1. **Analysis:** the goal of this phase is to elicit and describe system requirements in order to provide the best SW development within the system architecture. Impact analysis and mapping of functions into the defined SW architecture is performed in this phase.
2. **Design:** the goal of this phase is to identify the complete SW behaviour for each subsystem and the functions to be provided by each component. This level of refinement must include enough details in order to allow the subsequent coding phase.
3. **Implementation:** this phase is structured in the following sub-phases:
 - 3.1 *Coding and module testing:* with the goal to translate the design information into source code files using the defined computer language and to check the syntax and the semantic correctness of each source file.
 - 3.2 *Off-line testing,* with the goal to execute module testing in a simulated environment for the new/modified functions and to carry out non-regression testing for the unchanged functions with respect to the previous release.

4. **Test design and development**, with the goal to define the testing strategy and methodology, to design the tests and to prepare the environment for the relevant testing phases.
5. **Integration testing**: the phase is structured in the following activities:
 - 5.1 *Entity testing*, with the goal to carry out entity functional testing on target for the new/modified functions and to carry out non-regression testing for the unchanged functions with respect to the previous release.
 - 5.2 *Black Box - SBS Integration testing*, with the goal to test on target environment the old and new functions with all network elements (BTS, BSC, TRAU, MSC, OMC, SGSN).
6. **System Test**: the goal of this phase is: system validation (definition and execution of system functional tests and verification of compliance with the project specifications) of SBS base stations subsystems; design and execution of acceptance tests with designated customers for GSM products; technical support to customer acceptance testing executed by other organisational departments; technical support for designated networks and continuous strengthening and widening of the internal know-how for the reference areas.
7. **Maintenance**: the goal of this phase is to perform the SW product modifications due to correction of defects arisen after the product delivery or due to improvement performance of the product. During this phase the SW product is installed in the real operational environment, where the maintenance operations are performed through the delivery of new SW Loads or, exceptionally, Object patches.

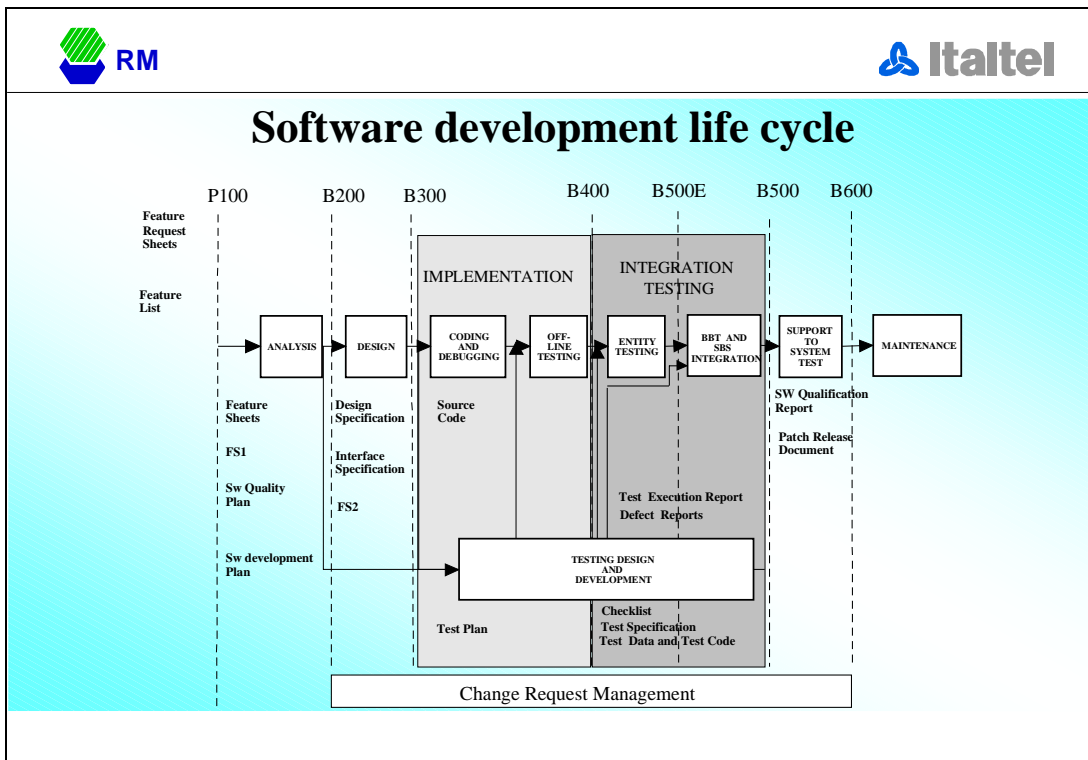


Figure 3.1 – Software development life cycle

3.2 TESTING ACTIVITIES

Concerning testing, besides review activities performed during the development phases, the validation of GSM systems involves several complex and effort-intensive tasks, that can be summarized as follows:

- **Module testing** performed by developers with the aim to have a first informal Test of the source code.
- **Off-line (host) testing:** performed by developers on stand-alone modules (or, when applicable, groups of modules) in a simulated environment.
- **Entity testing:** performed by developers following a feature oriented approach. Each feature chief with the support of the designers of the impacted functional areas verifies on target environment the whole feature using formal checklists and test specifications.
- **Black-Box - SBS Integration testing:** verification and validation (V&V) of a complete Network Element at the external interfaces in the target environment as well as interworking of interconnected Network Elements in fully equipped configuration.
- **System test:** V&V of the global system in the final environment, with an end-user perspective, including acceptance with customer.

At each step regression activities have to be performed with respect to features delivered in previous releases, features delivered in previous loads of the release under development, stability of the system after fixing of faults and/ or implementation of *Change Requests*, changes in hardware/ firmware / operating system/ configurations etc. As a consequence, regression testing has to be thorough, requiring considerable effort. Moreover regression testing is subjected to severe deadline pressures: testing is by definition on the critical path! The following picture summarises test activities.

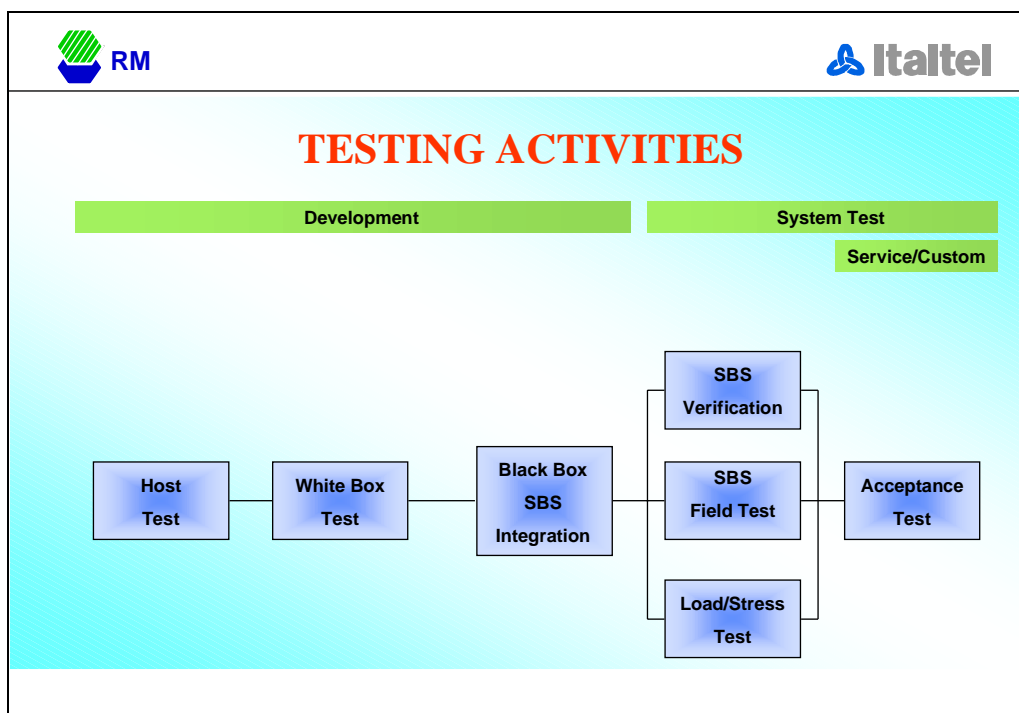


Figure 3.2 – Testing Activities

The test automation efforts described in the reminder of the paper focused especially on:

- Off line testing;
- Black Box - SBS Integration Testing.

4 TEST AUTOMATION IN THE CONTEXT OF SOFTWARE PROCESS IMPROVEMENT

The test automation experiences described in this paper are part of a larger Software Process Improvement (SPI) program that Italtel Reti Mobili is undergoing since 1995, with the following high-level goals [1]:

- to optimize the predictability of schedules;
- to further enhance product quality;
- to raise the availability and usability of documentation;
- to better the tool support;
- to keep/ increase productivity levels.

Italtel RM-RD is strongly committed to Software Process Improvement that is felt as a major leverage to increase the company capabilities; this is motivated by: the high world-wide competitiveness in the target domain, the increasing complexity of the software embedded in the delivered products and systems, the fact that projects are developed on an international multi-site basis and the increasing requirements from customers that are more and more demanding on software process maturity and stability.

The SPI program started in May-June 1995 with a Software Process Assessment conducted by Siemens Central Research, Application Centre Software. The assessment highlighted a good maturity level for the software producing unit, given that SPI at Italtel RM-RD was already an established practice. The SPI Program is intended as a continuous effort, handled with a management-by-objective approach with milestones and quantitative results.

In order to ensure its success, the PI (Process Improvement) Project has been organized as Figure 4.1 shows:

- a *PI Steering Committee* (referred in the following as “**PISC**”), chaired by the R&D Director and including all the managers reporting to him. The aim of this board is to define priorities, assign resources, solve problems and track the success of the initiative;
- a *PI Project Office* (called “**PIPO**” and equivalent to a SEPG), composed of a few experts, having the goal of planning/ tracking the project, giving technical guidance and harmonizing/ deploying the outcomes of the Working Groups; the PIPO has also the duty to organize the so-called “accompanying actions”, namely: training, dissemination and quantitative measurement.
- a number of *Working Groups*, composed of technical representatives from the various projects involved and dealing with improvement actions.

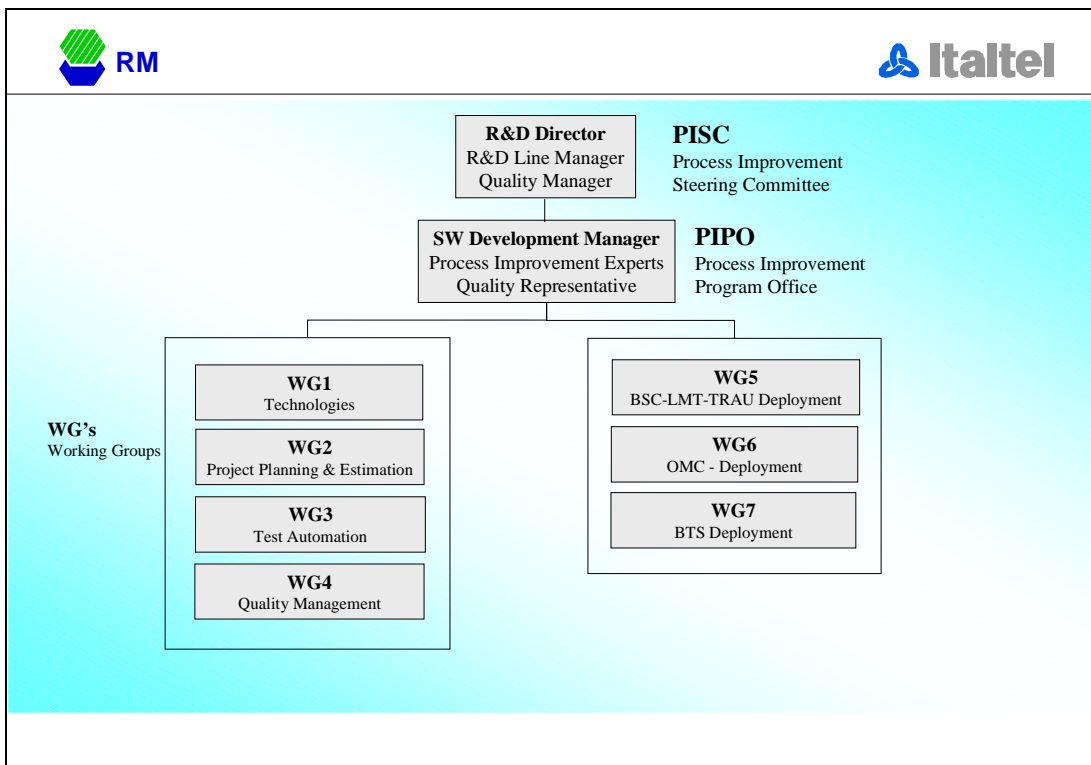


Figure 4.1 – SPI Program Organigram

Working groups are subdivided in two areas: the former (WG1 to WG4) with the goal to innovate that is to define/ enhance software engineering practices, the latter to deploy that is to apply defined practices, into development projects. The Software Process Improvement goals planned for 1999 as far as “innovative” WG’s are concerned are briefly summarised in the following table.

| | |
|------------|--|
| WG1 | <ol style="list-style-type: none"> 1. To integrate Configuration Management Environment between geographically dispersed development sites 2. To enhance Web interface for Project handling 3. Intranet services evolution 4. Trials for usage of formal specific languages 5. New technology watch for state-of-the-art technologies |
| WG2 | <ol style="list-style-type: none"> 1. To introduce new tools for effort tracking 2. To build estimation models based on historical quantitative data |
| WG3 | <ol style="list-style-type: none"> 1. To fine tune proprietary test automation tools 2. To evolve the existing tools for GPRS needs |
| WG4 | <ol style="list-style-type: none"> 1. To spread usage of quality indicators 2. To optimise SW development guidelines at R&D level |

Activities described in the remainder of this paper are related to goal 2 of WG3. The whole Test Automation environment is described in [2].

5 DATA TRANSMISSION ON MOBILE NETWORKS

5.1 HSCSD AND GPRS

Although the GSM commercial service started in 1992 exclusively for *Voice Services*, it was already fit for the purpose of meeting the increasing demand of *Data Transmission Services*. The GSM network, working in *Circuit Switched mode* (the network gives the customer the exclusive use of a certain amount of bandwidth for the duration of the call), allows to reach up to 12 kbit/sec of net data rate using one radio channel of 9,6 kbit/sec data rate. The connection is set up on demand and released when the caller hangs up.

Today, larger bandwidth is provided by combining more radio channels using *HSCSD (High Speed Circuit Switched Data)*.

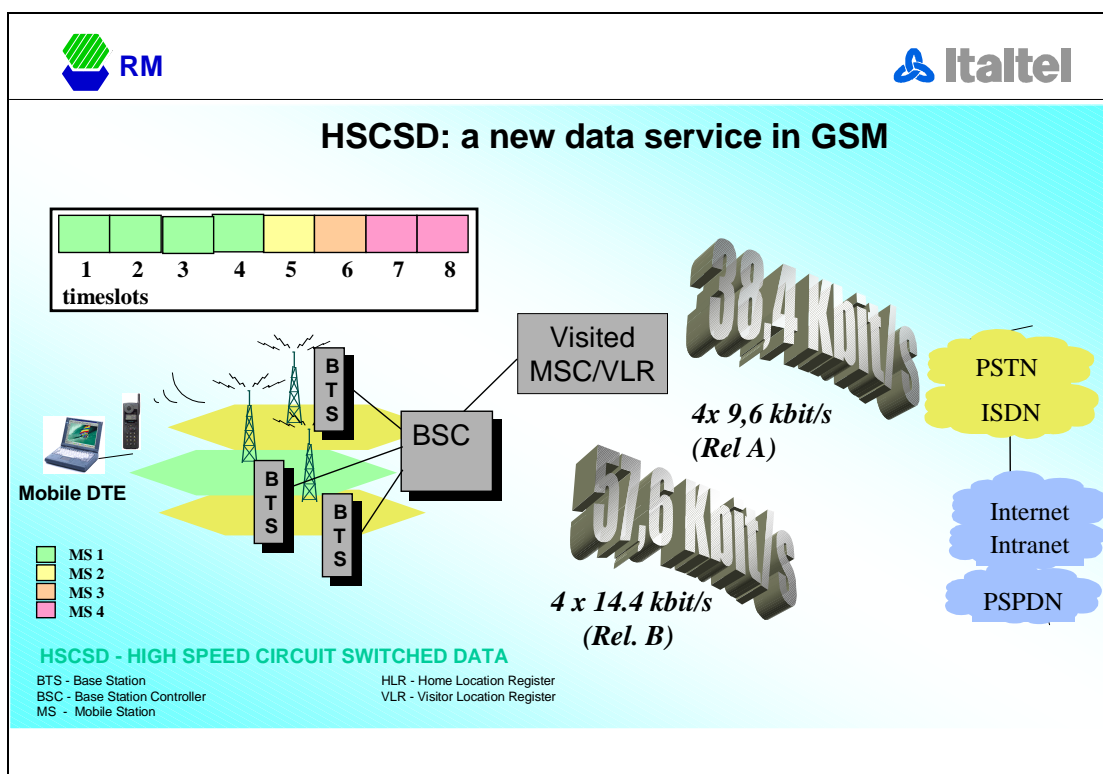


Figure 5.1 – HSCSD (High Speed Circuit Switched Data)

HSCSD is a *connection-oriented* data service (only point to point) for applications with high bandwidth demands and continuous data stream, e.g. motion pictures or video telephony. The higher bandwidth is achieved by combining 1 to 8 physical channels of one bearer frequency for just one subscriber. Additionally, the *coding schemes* allow a maximum of 14,4 kbit/sec instead of 9,6 kbit/sec to be transmitted per physical channel. In this way, HSCSD theoretically enables transmission rates up to 115,2 kbit/sec. In order to implement HSCSD, merely the GSM-PLMN (Public Land Mobile Network) software must be modified. More problematic is the high volume of resources needed.

If a particular user has nothing to send, which is typical of data communication, his/ her resource is wasted because it is not available to any other user. As a consequence, Circuit Switched connections do not provide an efficient mean to support data traffic, as they do not match the bursty nature of data communication.

As a further evolution, the main objectives to be reached by implementing GPRS are the following:

- to give support for bursty traffic;
- to use efficiently network and radio resources;
- to provide flexible services at relatively low costs;
- to allow high speed connectivity to the Internet;
- to provide fast access time;
- to have and support flexible co-existence with GSM voice.

As opposed to HSCSD, GPRS is a *packed-oriented* bearer service, meaning that the same radio channels can be shared by different subscribers. GPRS is a *resource-protective* procedure to implement applications with a short-term need for high data rates (e.g. surfing the internet, E-mail, ...). GPRS also enables broadcast transmission (point to multipoint) and charging based on the actual amount of data transmitted instead of the duration of the call.

In order to meet these objectives GPRS uses a packet-mode technique to transfer data and signalling in a cost-efficient manner over GSM radio networks and also optimizes the use of radio and network resources. Still a strict separation between the radio and network subsystems is maintained, in order to allow the network subsystems to be reused with other radio access technologies.

New GPRS radio channels are also defined. The allocation of these channels is flexible, ranging from one to eight radio interface timeslots per TDMA (Time Division Multiple Access) frame, and they can be shared by active users. The introduction of new coding schemes with transmission rates of up to 21,4 kbit/sec per physical channel enable theoretical transmission rates up to 171,2 kbit/sec. With such bit rates, all types of transmissions can be handled: from slow-speed short messages to the higher speeds needed e.g. when browsing Web pages. GPRS will also allow the user to receive voice calls simultaneously when sending or receiving data calls.

GPRS provides a seamless connection to the existing standard data services by using interfaces to TCP/IP and X.25. GPRS will also provide fast reservation to begin transmission of packets, typically from 0,5 to 1 second. This means that the data users will not have to wait for the phone to dial, but instead they will get through immediately. For example the messages will be delivered direct to the user's phone, without the need for a full end-to-end connection. When the user switches on his/ her phone, the messages are downloaded automatically.

5.2 GPRS System Architecture

GPRS introduces an *overlaying* architecture on the existing one with the definition of new entities and new interfaces as the Figure below shows.

These modifications concern:

- linking the BSS to the new Serving GPRS Support Nodes (**SGSN**) via Gb-interface, i.e. setting-up a packet control unit PCU;
- transmission of the **PDU** (**P**acket **D**ata **U**nit) via BSS;
- the new channel coding schemes, i.e. realising a GPRS Channel Codec Unit (**CCU**), as well as
- the possibility of combining physical channels with a view to achieving high transmission rates via the radio interface Um.

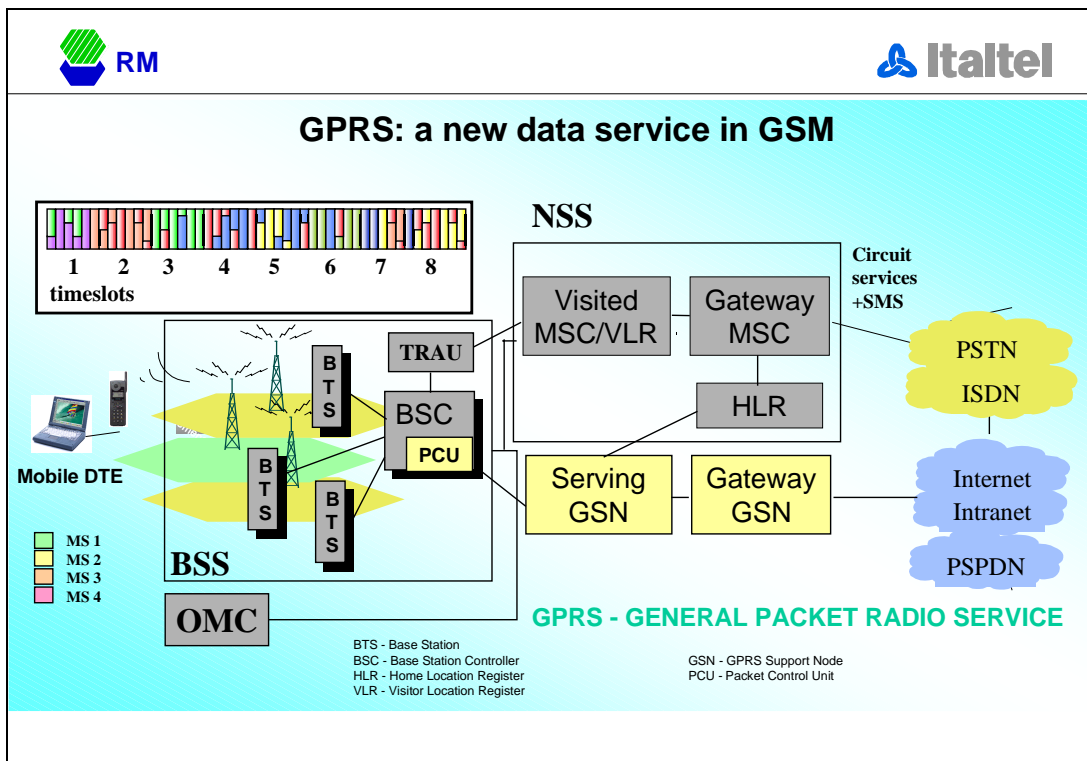


Figure 5.2 – GPRS Reference Model

As far as the BSS is concerned, the GPRS functional units are the **CCU** located in the BTS and the **PCU** located in the BSC.

Referring to the *Base Station Controller* (BSC) the network structure of the GPRS requires the introduction of a new interface in the direction of the SGSN-GGSN. This is due to the packet-oriented data transfer in the GPRS and the corresponding protocols, which now also need to be handled in the formerly connection-oriented BSS. In the SBS, this interface is obtained by extending the BSC with the packet control unit PCU. This is done by means of a modification in the BSC hardware which consists in inserting PCU cards into the BSC rack.

Multi-vendor capabilities are guaranteed by a standard Gb interface, while the Abis transport mechanism is a proprietary PCU-frame format. This is due to the fact that GSM has not completely specified the Abis interface, so that each manufacturer may choose a PCU format which is suitable to its own needs.

The figure below shows the protocol stack for GPRS network.

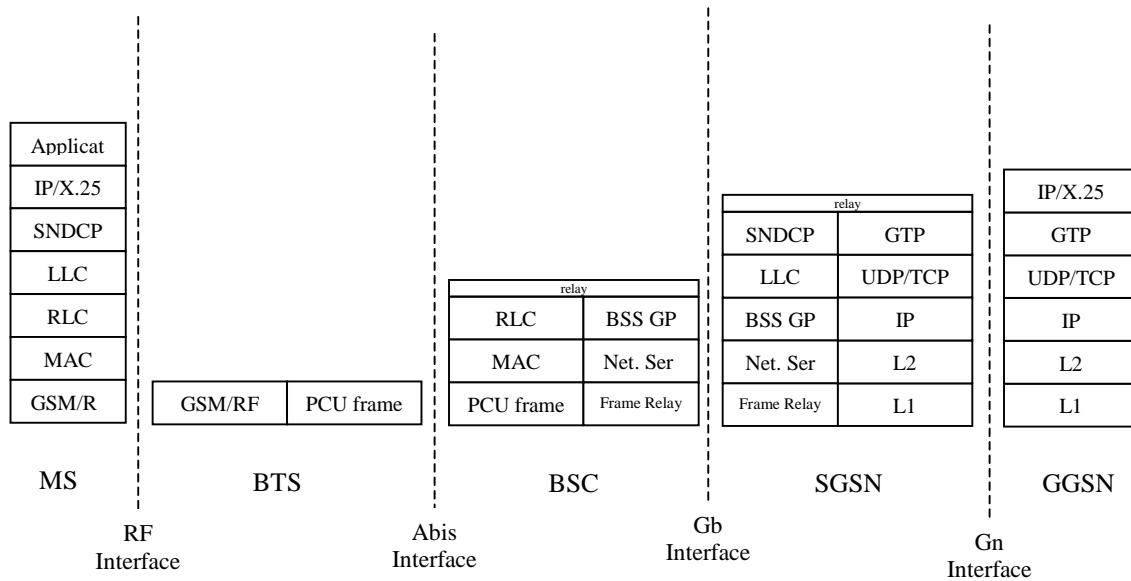


Figure 5.3- GPRS Protocol Stack

The *PCU* is a functional unit within the BSC whose task is to provide resource allocation and protocol conversion between BTS and SGSN. The PCU is responsible for:

- channel access control functions, e.g. access requests and grants;
- PDCH (Packet Data Channel) scheduling functions for uplink and downlink data transfer;
- radio Channel management functions, e.g. power control, congestion control, broadcast control information, etc.
- functions such as buffering and retransmission of data block;
- packet data unit segmentation for down link transmission;
- packet data unit re-assembly for up link transmission;

The functions inside the *CCU* are:

- channel coding functions;
- radio channel measurement functions including received quality level, received signal level and information related to timing advance.

PCU frame are transferred across the Abis interface.

The PCU acts as a BSC for statistical multiplexing and routing; in fact it receives RLC packets from the Abis channel related to more than one mobile and it packs them into LLC frames; these LLC frames are then routed, together with other LLC frames coming from other Abis channel to the SGSN, and vice-versa

6 TEST AUTOMATION ENVIRONMENT FOR GPRS TESTING

The rapid growth rate of the radio mobile market drives continuously to an equal evolution of the network equipment needed to provide the radio mobile services and the techniques/ tools for their development and testing. Using flexible tools, able to execute a great number of tests, to provide all the services needed by the network elements, it is expected to reduce the test effort/ schedules improving at the same time the reliability of release and product.

The set-up of the test Automation environment started with a feasibility study, which had the goal to define the architecture of the overall environment and its components. Afterwards, a market analysis was conducted to identify whether commercial tools existed to cover the needs or an internal development had to take place.

This allowed to define a road-map for the Test Automation Project, including the development of proprietary solutions, the set-up of the underpinning infrastructure, the procurement and validation of most promising commercial tools and their piloting in case studies. In particular, referring to the GPRS feature testing, an high level view of the Test Automation environment is given in Figure 6.1

Besides the Network Elements, which are described in earlier parts of this document, the following test tools are identified:

HATT (Host Automation Test Tool): a proprietary tool dedicated to the host testing phase that automatically executes a sequence of test drivers.

LSU Plus (Line Server Unit plus): a multiprocessor system with HW and SW developed ad hoc to satisfy the testing of complex telecommunications equipments such as the Base Station Controller in Circuit Switched and GPRS modality.

GTAS (GSM Test Automation System): a proprietary tool, based on client/server architecture, which allows and supports automatic and synchronized test execution for a set of GSM network tools.

Abis Applications

- 1- *LabLog*: it is a monitor tool of the messages at the level of PCU frames.
- 2- *LabRes*: it allows the sending and the check of received packet telephonic procedure messages driven by script files.
- 3- *LGTS Plus*: it allows to simulate a traffic scenario with both Circuit Switch (CS) and GPRS mobile stations.

Gb Applications

1. *GbLog* it is monitor tool of the messages at any level of protocol stack; Frame Relay (**FR**), Network Services (**NS**) or BSSGP packet data unit (**PDU**) are decoded and logged.
2. *GbSeq* allows the sending of messages interactively, driven by user or by file: the interface is permitted at any level (FR, NS, BSSGP).
3. *GbTraffic*: it is a tool providing a loop path for the signaling and data traffic generated and monitored by the Abis simulator on MS couples.

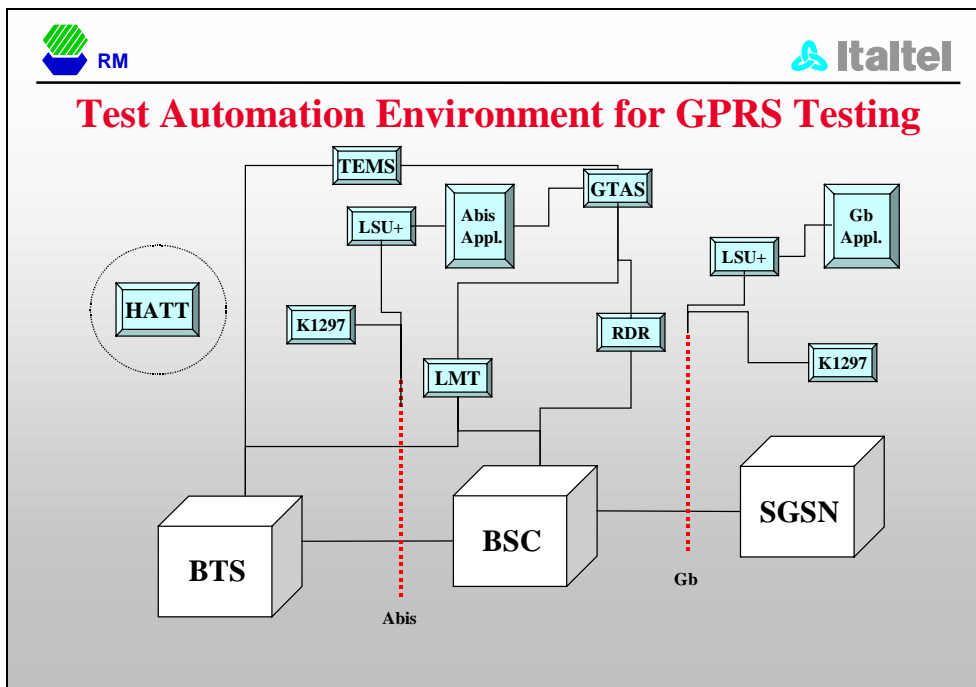


Figure 6.1 –Test Automation environment for GPRS

Commercial tools

1. K1297: it is a commercial Multiprocessor/ Multi-interface Tester for monitoring, simulation/emulation and conformance tests.
2. TEMS (Test Equipment Mobile Station): a commercial tool that allows the programming and control of phone calls.
3. RDR (Relay Driver): it is a server which can control (open/ close) electric circuits in order to generate faults.

The following paragraphs describe the tools developed within the Test Automation Project.

6.1 HATT (HOST AUTOMATION TEST TOOL)

The HATT performs tests in automatic way on simulated environment, covering both execution of tests and checking of their results. It interprets a simple language that allows:

- to send, receive and trace messages among tasks;
- to perform the function calls and the function results manipulation;
- to give some simple control structure;
- to check and update global variable values;
- to interact with the object of the operating system, for example semaphores.

HATT automatically executes a sequence of test drivers listed in a file (*Test Chain File*). In every line of this file there is the name of another file (*Test Script File*) which contains a list of commands used for:

- sending, receiving and checking messages;
- setting names of logfiles;
- accessing processor memory;
- interacting with stubs;
- etc...

These commands are interpreted and automatically executed. The result of the test depends on the result of every command in the test script and it is automatically checked. When a test is executed (successfully or unsuccessfully) the tool starts to interpret the next Test Script File in the list of the Test Chain File unless there is an indication to execute an initialization test or another test script. Message flows are traced and the message contents recorded in logfiles.

6.2 LSU Plus (LINE SERVER UNIT PLUS)

The LSU Plus is a multiprocessor system with HW and SW developed ad hoc to meet the test requirements of complex telecommunications equipment such as the Base Station Controller (BSC).

The LSU Plus can be connected to any of the various interfaces of a GSM network element (i.e. Abis, Asub, A, O, Gb) through eight PCM E1/T1 lines and the support of up to 256 serial channels (8-2048 Kb/s) that allow HDLC based signalling or others special frame formats (PCU Frame, TRAU Frame, V.110). A typical connection for the GPRS testing is shown in figure 6.2.

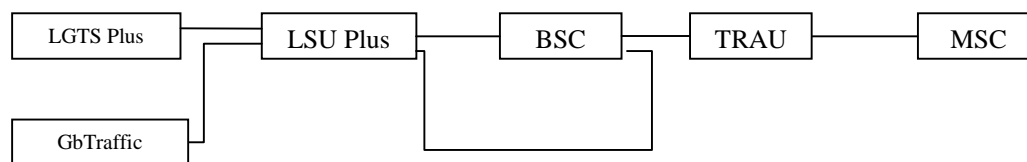


Figure 6.2 – A sample test line for GPRS

A switch matrix allows to set-up the connections between the time slots of the PCM lines and the serial channels

Referring to the test of the GPRS feature, the LSU Plus is able to handle the protocol stack at both the Abis (RLC/MAC, LLC ...) and the Gb interface (Frame Relay, Network Service BSSGP ...).

Moreover the LSU Plus includes the LAPD and MTP protocol layers, satellite delay simulation, as well as some special facilities for testing circuit switched speech and data channels.

The single protocol layers can operate in monitor mode or as active peer. Concurrent logging of multiple channels with different protocol in a single stream is supported.

The system is the base building block for developing simulators of network elements oriented to the various stages of the software process development and to the system test.

6.3 GTAS (GSM TEST AUTOMATION SYSTEM)

GTAS is a proprietary tool, based on client/server architecture, which allows and supports automatic and synchronized test execution for a set of GSM network elements.

The test environment is composed of several devices connected through a local area network and communicating via Ethernet using TCP/IP protocol (Figure 6.3). GTAS station allows a centralized access to such devices and manages the synchronized execution of commands and retrieval of results. The role of the GTAS main process is to execute test scripts, constituted of instructions for sending inputs and for checking of system reactions as well as to collect and log test results, in order to allow subsequent evaluation of results themselves.

The GTAS main features are implemented by a set of functional modules:

a *GTAS main process*, able to execute test programs and to log test results;
 a *set of Agents*, one for each server, possibly running on the device machine, which know the specific characteristics of the device and act as remote executor for GTAS;

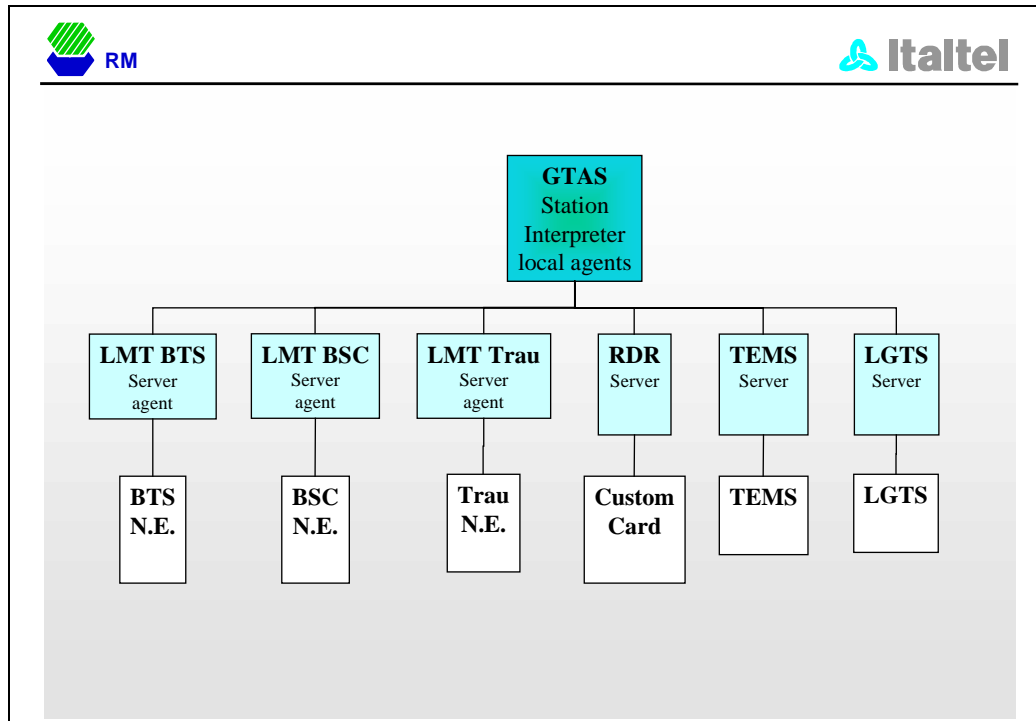


Figure 6.3 – GTAS architecture

an *user interface module*, which allows the user to start and stop the test, as well as to verify the execution status of the test itself (considering that also the single script could be long lasting);

a *Common Interprocess Communication Module (CICM)*, which supports the message exchange between the above functional modules

GTAS functionalities are represented by suitable objects based on RTO (Real Time Object) architecture:

one object represents the test program interpreter;

a set of objects (LocalAgents), one for each Agent, represent the connections with the Agent themselves, holding the connection status and translating protocol messages into internal messages and method invocation;

one object manages the connection with the user interface process and translates external messages into internal commands;

one object manages the logging of commands and answers in a suitable file structure, supplying a time stamp and a Agent identifier, in order to allow the retrieval and selection of test events by time interval and source or destination;

one object represents the overall status of the GTAS main process and of the test program under execution, in order to supply to the user interface process all necessary information to make the user aware about the test progress.

The GTAS user specifies the sequence of commands to be submitted to the actual devices by a suitable high level language, allowing to send a command to the proper device and to

synchronize the execution of the subsequent command with the receipt of some particular messages from the same or another device.

6.4 Abis Applications

6.4.1 LABLOG

It is a monitor tool of the messages at the level of PCU frames. It is able to test the correctness of the PCU frames flowing through the interface linking the BSC and the BTS or its simulator; it is also able to decode in clear text the in-band signaling messages on this interface on one or more 16 Kbps channels and monitor them on the PC screen and/ or log them in a binary formatted file.

6.4.2 LABRES

It is an application based on the LGTS (LSU- Based GSM Traffic simulator) software to which some new high level definition language instructions and a GUI has been added.

The LSU Plus offer a pool of primitives to interface the RLC/MAC level to obtain the control of all telephonic procedures and LLC frames sending which LabRes uses to debug one GPRS procedure through user scripts and to generate concurrently CS traffic.

High definition language (HDL) is enhanced with the instructions needed to control, with a C-like syntax, all type of GPRS procedures at a message level: variables and types definition, expressions and flow control instructions are available.

A graphic application and the insertion of compiler and debugger functionality also improve the user interface: displaying of HDL programs and user input commands.

6.4.3 LGTS PLUS (LINE SERVER UNIT GSM BASED TRAFFIC SIMULATOR PLUS)

The LGTS Plus is a programmable telephonic traffic simulator able to generate a pre-definite traffic level on the Abis interface. The high level of programmability makes the LGTS Plus a very flexible tool able to manage most of the BSC testing requirements. Given that at the time being there does not exist a reference traffic model for GPRS, one of the main requirement for testing is to study the BSC behaviour corresponding to:

- traffic changes models;
- traffic intensity corresponding to a fixed traffic model;
- cells configuration.

The LGTS Plus provides:

- a programming language useful to describe the Call processing procedures referring either to the Circuit Switch mode or the GPRS mode;
- a language able to describe traffic scenarios (a traffic scenario is a set of Mobile Stations, and Call Processing procedures with respective activation frequency);
- an integrated environment able to simulate at the same time the traditional MS+BTS behaviour, the MS+BTS supporting the HSCSD behaviour and the MS+BTS that support the GPRS behaviour;
- an appropriate report about the traffic level load at real time by the network
- a connection to the GTAS tool to automate test execution
- an appropriate logging of all the messages as well as a set of utilities for post-processing useful to analyse how the test is evolving and to understand whether errors have occurred.

From a functional point of view, the LGTS Plus can be broken-down into the following modules:

User Interface (UI): it provides to the user all the commands needed to interact with the simulator;

Supervisor: it manages all the user commands;

Translator: it translates the programs and the procedures described by the user in a low level format, used by the Call processing sub simulator;

Call Processing (CP) Simulator: it is the kernel of the LGTS Plus; it manages the Call processing simulation using the scenarios described by the user and providing all the functionalities needed to generate the run-time reports;

Operation and Maintenance Simulator (O&M): it is able to simulate the alignment of the BTS used in the test session, from a O&M point of view, providing to the CP simulator all the BTS parameters needed to the CP simulation.

Dispatcher and Sender (D&S): it manages the communication with the LSU Plus and optionally with the GTAS tool via TCP/IP sockets.

Figure 6.4 shows a possible Call Processing scenario that can be simulated by the LGTS Plus.

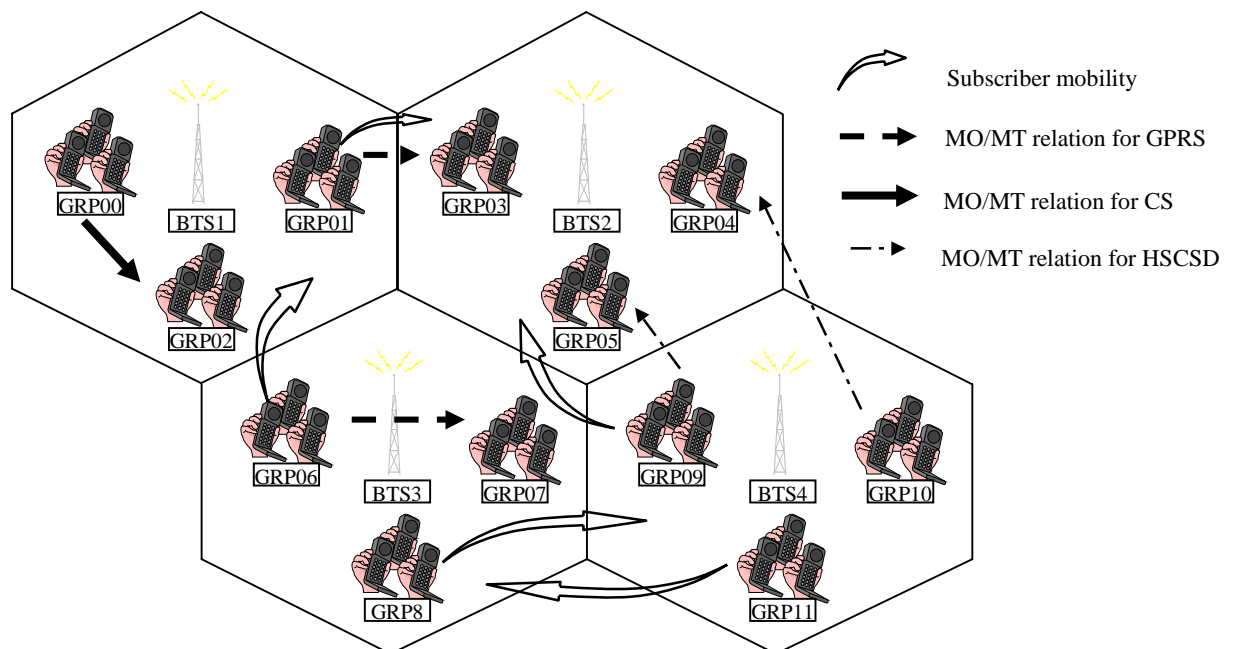


Figure 6.4 – An example of Call Processing Scenario

GRPXY: it represents a MS's group that shares the same behaviour (i.e. performs a Mobile Originated Packet Data Transfer Procedure)

BTSX: is a BTS that represents a specific cell with appropriate parameters (i.e. a specified number of TRX with a given number of signalling and traffic channels).

The LGTS plus allows to describe and to simulate the subscriber activities and mobility as the following tables shown.

| Subscribers activity | |
|----------------------|--|
| CS | Relation between Mobile Originated Call and Mobile Terminated Call Mobile Originated, Mobile Terminated |
| HSCSD | Relation between Mobile Originated Call and Mobile Terminated Call Transparent or not transparent Mobile Originated Call for every mobile multislot class Transparent or not transparent Mobile Terminated Call for every mobile multislot class |
| GPRS | Relation between Mobile Originated and Mobile Terminated Packet Data Transfer Data quantity to transfer with a fixed QoS |

| Subscribers mobility | |
|----------------------|--|
| CS+HSCSD | Location update Handover |
| GPRS | Cell Updating Routing area updating Cell reselection |

From a BSC point of view the traffic models that can be simulated by the LGTS Plus are explained in the following:

$$TM = TCS + THSCSD + TGPRS$$

Where:

$$TCS = aMOC + bMTC + cHO + dLU \text{ (with a connection time fixed)}$$

$$THSCSD = \sum_{i=1}^n \sum_{j=1}^{19} (a_i MOCT(MsCl = x_j) + b_i MOCnT(MsCl = x_j) + c_i MTCHS(MsCl = x_j) + d_i HOT(MsCl = x_j) + e_i HOnT(MsCl = x_j))$$

(with a connection time fixed)

$$TGPRS = \sum_{i=1}^n \sum_{j=1}^{29} \sum_{k=1}^4 (a_i MOPDT(PDUs = x_i, MsCl = x_j, QoS = x_k) + b_i MTPDT(PDUs = x_i, MsCl = x_j, QoS = x_k) + c_i CR(PDUs = x_i, MsCl = x_j, QoS = x_k) + d_i CU + e_i RU)$$

TM = Traffic Model

TCS = Traffic model for Circuit Switch

THSCSD = Traffic model for HSCSD

TGPRS = Traffic model for GPRS

MOC = Mobile Originated Call

MTC = Mobile Terminated Call

HO = Handover

LU = Location Update

MOCT = Mobile Originated Call Transparent (for HSCSD)

MOCnT = Mobile Originated Call not Transparent (HSCSD)

MTCHS = Mobile Terminated Call (HSCSD)

HOnT = Handover referring to a not Transparent call

MOPDT = Mobile Originated Packet Data Transfer

MTPDT = Mobile Terminated Packet Data Transfer

CR = Cell Reselection

CU = Cell Updating

RU = Routing Area updating

MsCl = Multislotclass

PDUs = Packet Data Unit size;

6.5 Gb Applications

6.5.1 GBLOG

It is a monitor tool of the messages at any level of protocol stack; Frame Relay (**FR**), Network Services (**NS**) or BSSGP packet data unit (**PDU**) are decoded and logged. It is able to decode in clear text any message of the three levels of this interface and monitor them on the PC screen and/ or log them in a binary formatted file with time stamp. The file, can be then post processed and filtered to search for items of interest, basing on the protocol level and on some predefined categories; another message flow can be produced on the screen or on another file.

6.5.2 GBSEQ

GbSeq allows the sending of messages interactively: the interface is allowed at any level (FR, NS, BSSGP); it is a client applications of the LSU Plus, like the *GbLog*, but moreover it allows to send messages on the Gb interface towards the BSC. It can be used to simulate, at any level, the Gb protocol stack on the BSC (playing the network side role), handling different links at different protocol layers and at different independent rates. The files of messages are treated in different ways: to send a message at a time, all messages with specified delays in between, restarted from the beginning.

6.5.3 GBTRAFFIC

The GbTraffic is a simulator able to simulate, from a BSC point of view, the SGSN behaviour. It is based on the services provided by the LSU Plus and works in close co-operation with the LGTS Plus. The GbTraffic is able to handle the signalling and data generated on Gb interface from LGTS Plus traffic simulator.

The main features of the Gb traffic simulator are the following:

- to handle the relation between Mobile Originated and Mobile Terminated packet transfer (MOC/MTC);
- to handle the packet data transfer between two related MS or between a MS and the Network;
- to handle subscriber mobility;
- to provide an appropriate report on the simulation;
- to provide a language to configure the LSU Plus protocol layers

From a BSC testing point of view, both the subscriber mobility management and the MO/MT relation management can be simplified, at configuration time, at some cells and the relation MO/MT is fixed at configuration time. For example if an MS performs a cell reselection from BTS1 to BTS2 (see figure 6.5) in the real network then the SGSN will page the MS in the new cell after the procedure has been completed; in the simulated network the traffic simulator will page the MS always in the same starting cell (also after the cell reselection).

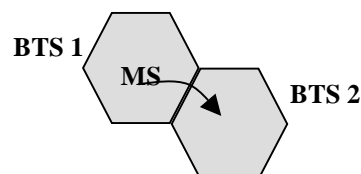


Figure 6.5- Cell reselection

7 PRELIMINARY RESULTS AND THE WAY AHEAD

Preliminary results show that, in the areas where test automation has already been deployed:
 test productivity has improved (Figure 7.1);

fault density has improved (Figure 7.2).

This is boosting additional efforts on test automation.

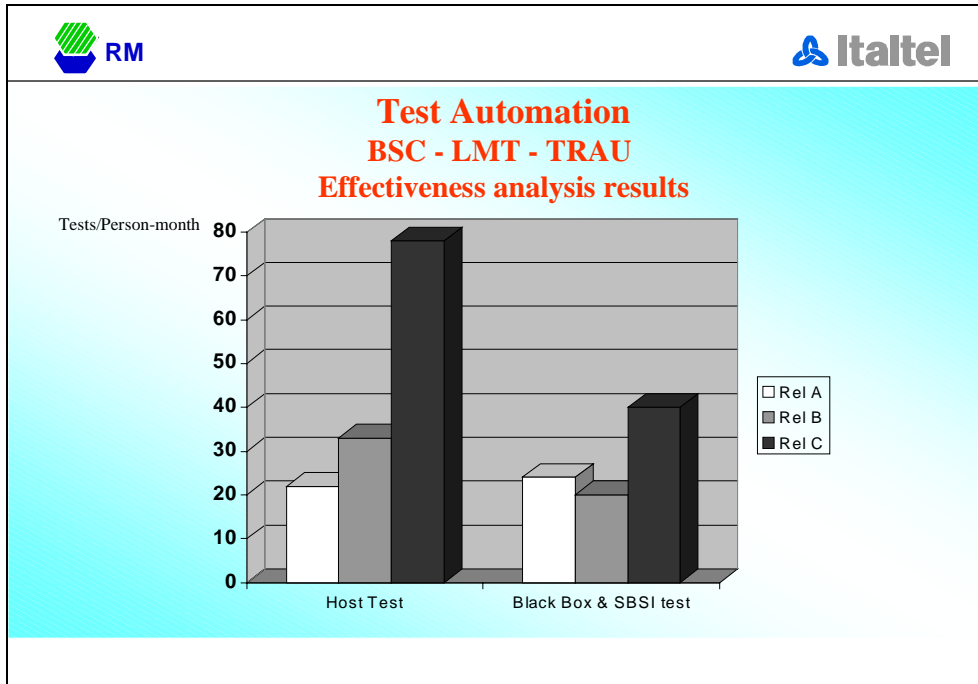


Figure 7.1 - Test Automation Effectiveness

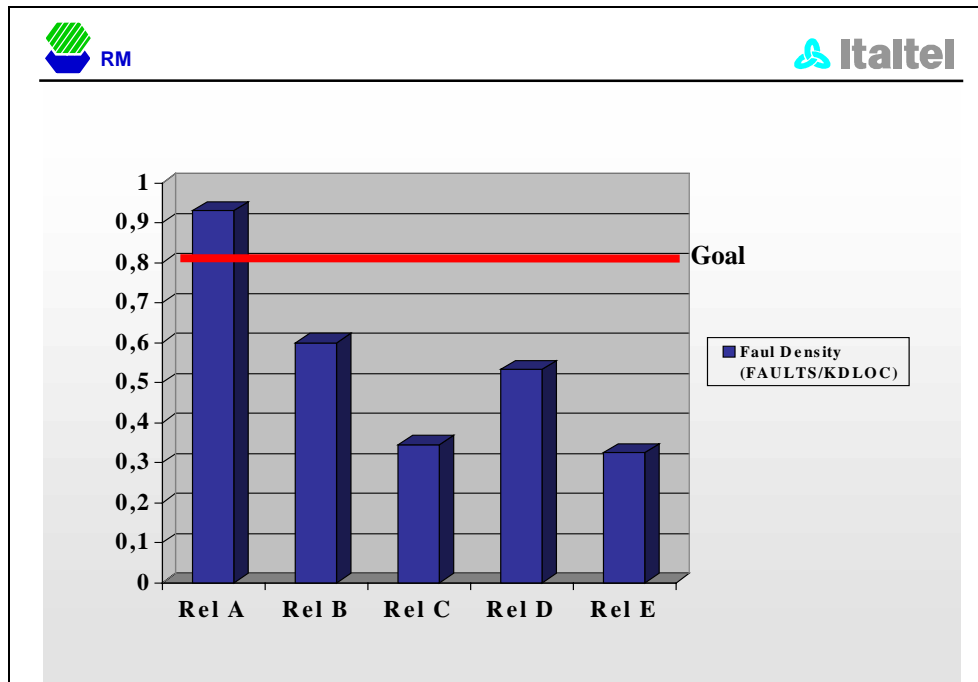


Figure 7.2 – Fault density

The next steps on our roadmap foresee at the time being:

- Field fine-tuning of all the applications till now developed
- Full deployment of tools according to user requirements
- Deployment of various tools to other sections of the SBS projects

The first point is going to lead to an extensive use of GTAS with the addition of other controlled tools. As it can be seen the future objectives are not only of technical nature, but also involve organizational topics. This is important in order to extend the application area of these techniques because previous analysis about test automation effectiveness [3] revealed that a considerable return on investment can be achieved.

Of great importance is the database organization for the tests developed, in terms both of script and environment description, so to ease the future reuse.

The quantitative tracking of software releases development of the whole project must be continued to understand whether the improvement direction is kept and to maintain an updated set of information useful for decision making.

ACKNOWLEDGEMENTS

Our thanks go to G. Cecchetto and S. Di Muro who supported and sponsored the initiative. The Test Automation project has been made possible by the people taking part in the technical activities, including: O. Cantoni, E. Colombo, D. Gurrieri, A. Ferrante, M. Mancini, C. Orlando, R. Palmer, G. Regattin, C. Serrelli.

REFERENCES

- [1] S. Di Muro, A. Lora, S. Scotto di Vettimo, G. Rumi - "*SPI: an Experience Report from GSM Development*" - AQUIS 98 Conference - Venice, March 1998
- [2] R. Delmiglio, G. Bazzana, L. Annoni, A. Manini, E. Parenti, C. Serrelli, C. Trevisson - "*Test Automation Experiences in Telecommunication Software*" - EUROSTAR 97 - "5th European Conference on Software Testing Analysis and review" - Edinburgh - November 1997
- [3] G. Bazzana, R. Delmiglio, A. Lora, S. Finetti, O. Balestrini "*Quantifying the benefits of software testing: an Experience Report from the GSM Application Domain*" - Proceedings of Objective Software Quality Conference - Florence, May 1995

3rd International Software Quality Week Europe

Test Automation in Telecommunications Software: a Case Study on GPRS

R. Delmiglio⁽¹⁾, A. Manini⁽¹⁾
G. Bazzana⁽²⁾ - G. Rumi⁽²⁾ - F. Basili⁽²⁾
E. Bendinelli⁽³⁾ - A. Rappelli⁽³⁾



[1] **Italtel SpA**
Reti Mobili Research and Development department
SS11, Km 158
20060 Cassina De Pecchi (Milan) – Italy

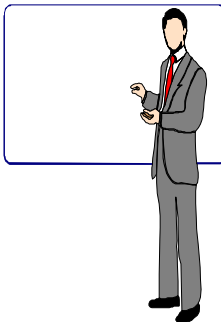


[2] **ONION SpA**
Communications, Technologies, Consulting
Via L. Gussalli, 9
25131 Brescia – Italy



[3] **SpazioZeroUno SpA**
Via Grande, 21
20090 Vimodrone (Milan) - Italy

Contents of the Presentation



Company Profile

Evolution of GSM and Mobile Telephony

Software Development and Testing Processes

Test Automation in SPI context

Data Transmission on Mobile Networks

Test Automation for GPRS Testing

Preliminary Results and the way ahead

Italtel Company Profile

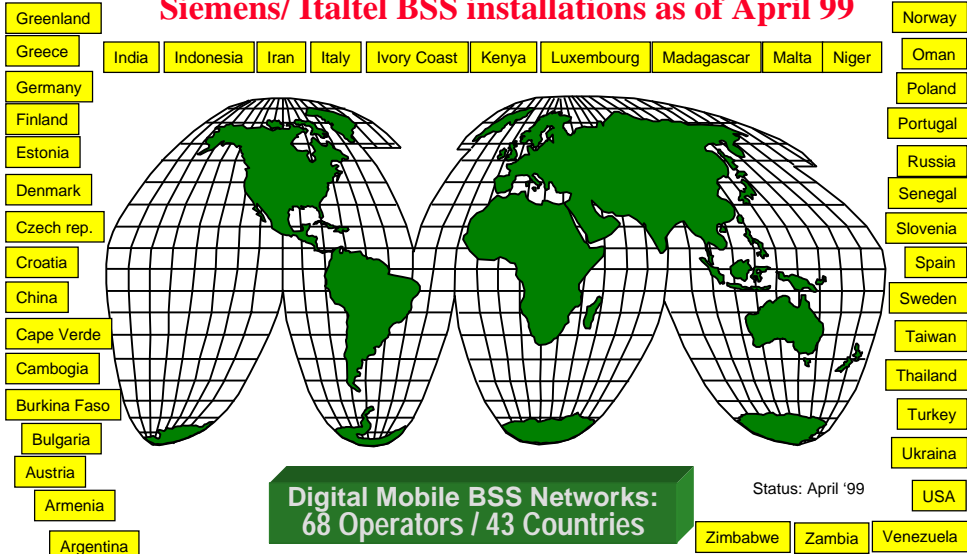


- Designs, manufactures, markets and installs systems and equipment for public and private applications;
- Main sector: Switching, Mobile Network, Transmission;
- Over 13% of sales invested in R&D.

Mobile Network Business Unit mission

- to be a turn key mobile network supplier;
- to develop the business of mobile networks;
- to be the competence centre for the whole Italtel-Siemens group for Base Station Subsystems;
- to support customers in all project implementation phases.

Siemens/ Italtel BSS installations as of April '99



Company Profile

Evolution of GSM and Mobile Telephony

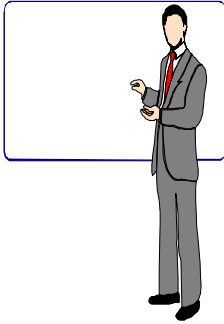
Software Development and Testing Processes

Test Automation in SPI context

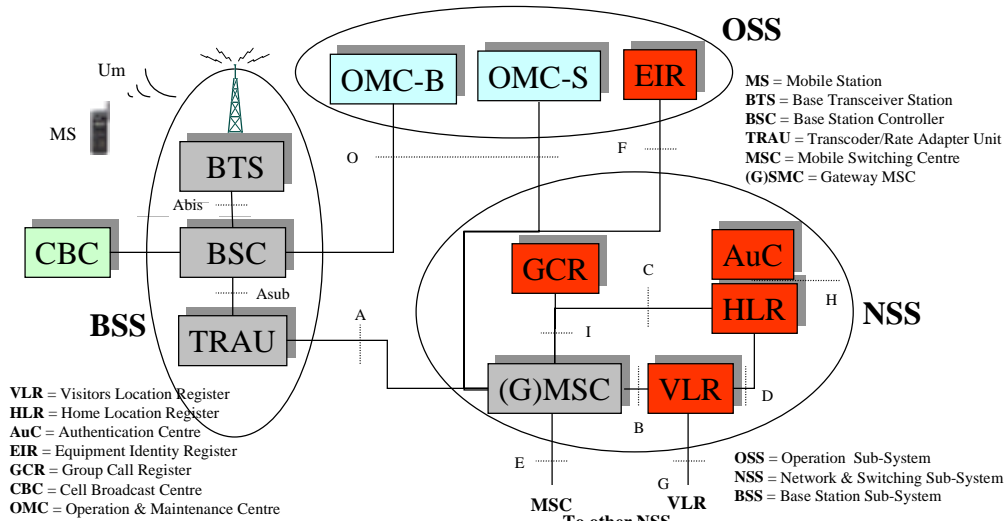
Data Transmission on Mobile Networks

Test Automation for GPRS Testing

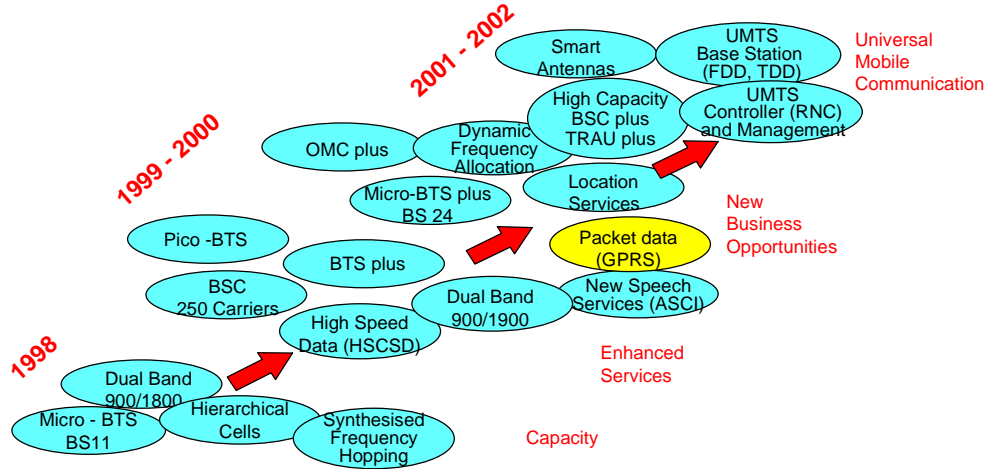
Preliminary Results and the way ahead



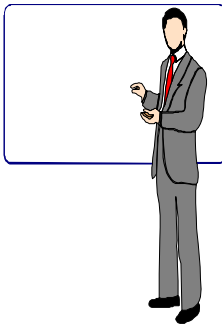
GSM Architecture



Italtel BSS Line Evolution



Italtel products / Services



Company Profile

Evolution of GSM and Mobile Telephony

Software Development and Testing Processes

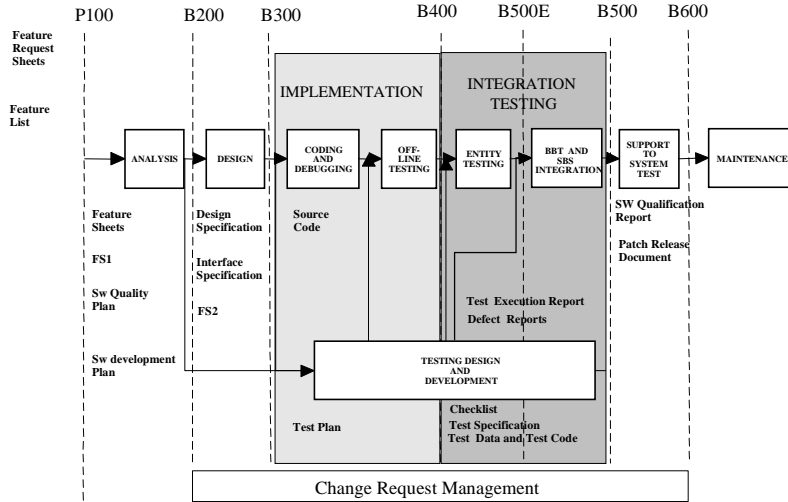
Test Automation in SPI context

Data Transmission on Mobile Networks

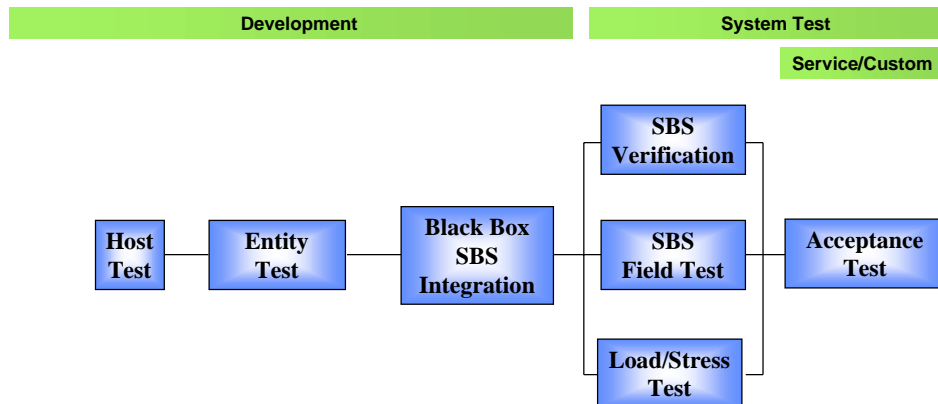
Test Automation for GPRS Testing

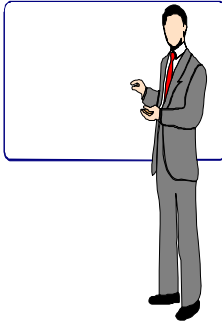
Preliminary Results and the way ahead

Software development life cycle



Testing Activities





Company Profile

Evolution of GSM and Mobile Telephony

Software Development and Testing Processes

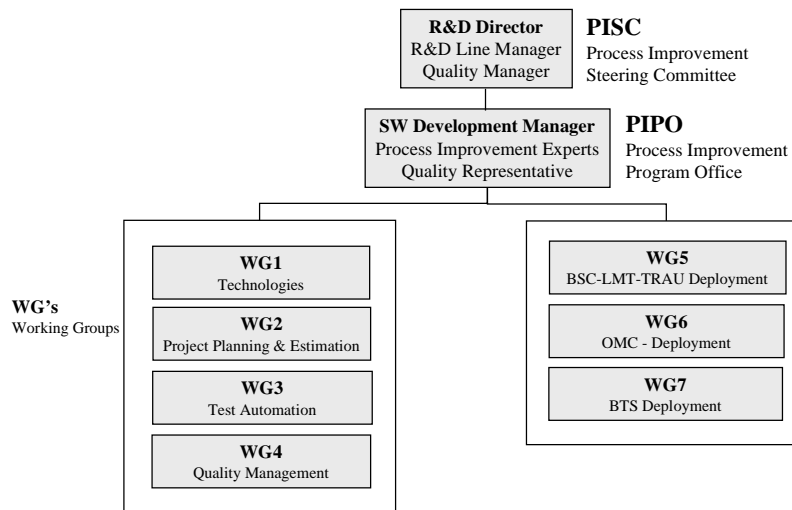
Test Automation in SPI context

Data Transmission on Mobile Networks

Test Automation for GPRS Testing

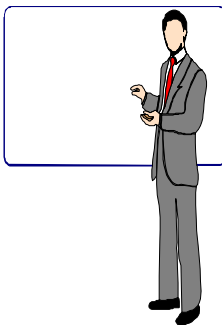
Preliminary Results and the way ahead

Software Process Improvement Initiative



Software Process Improvement goals for 1999

| | |
|------------|--|
| WG1 | <ol style="list-style-type: none"> 1. To integrate Configuration Management Environment between geographically dispersed development sites 2. To enhance Web interface for Project handling 3. Intranet services evolution 4. Trials for usage of formal specific languages 5. New technology watch for state-of-the-art technologies |
| WG2 | <ol style="list-style-type: none"> 1. To introduce new tools for effort tracking 2. To build estimation models based on historical quantitative data |
| WG3 | <ol style="list-style-type: none"> 1. To fine tune proprietary test automation tools 2. To evolve the existing tools for GPRS needs |
| WG4 | <ol style="list-style-type: none"> 1. To spread usage of quality indicators 2. To optimise SW development guidelines at R&D level |



Company Profile

Evolution of GSM and Mobile Telephony

Software Development and Testing Processes

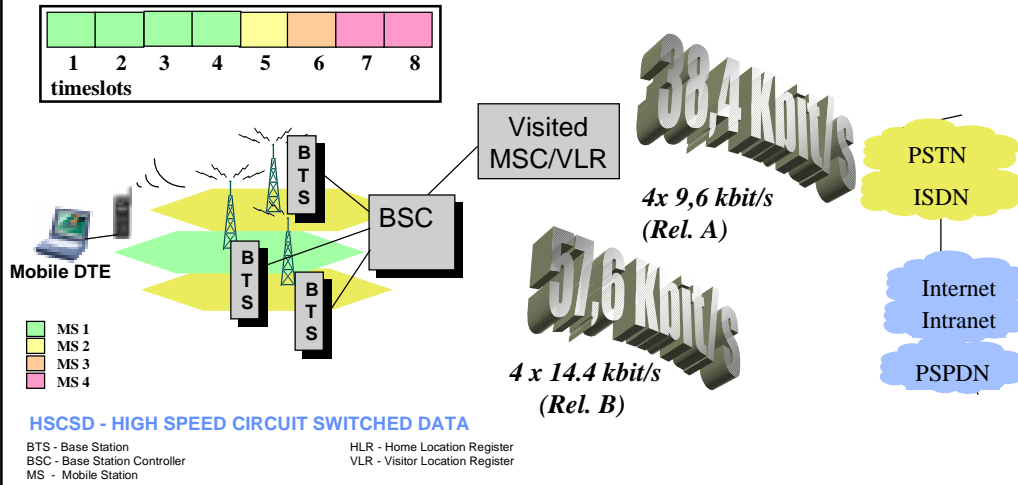
Test Automation in SPI context

Data Transmission on Mobile Networks

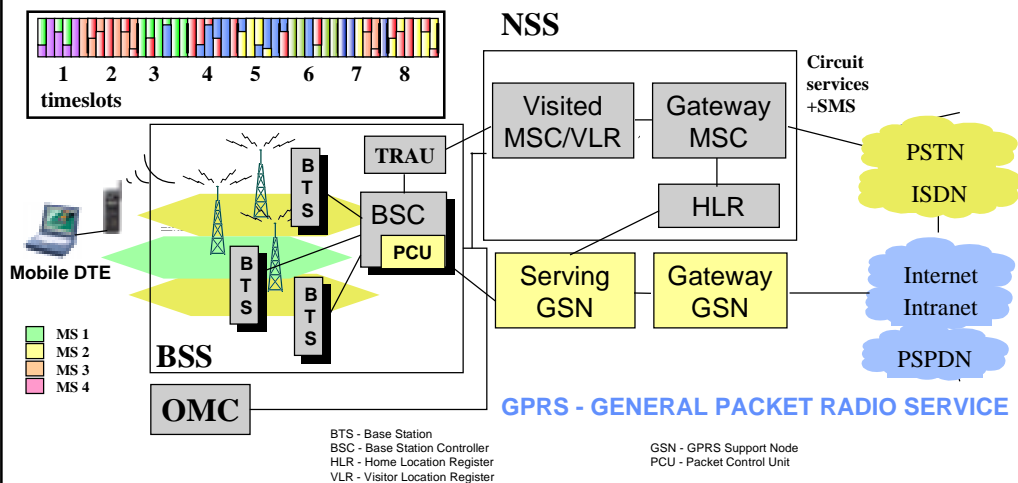
Test Automation for GPRS Testing

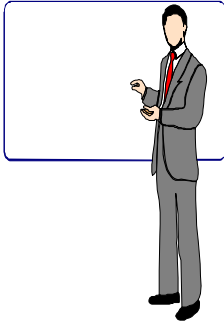
Preliminary Results and the way ahead

HSCSD: a new data service in GSM



GPRS: a new data service in GSM





Company Profile

Evolution of GSM and Mobile Telephony

Software Development and Testing Processes

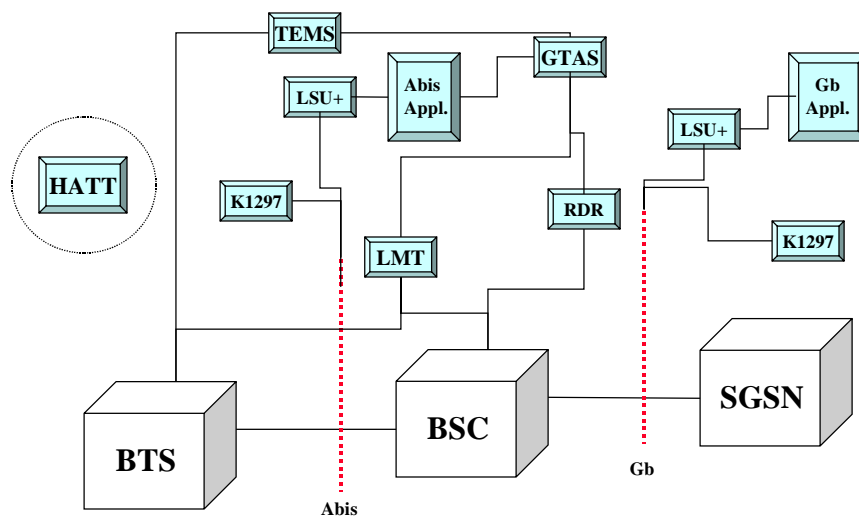
Test Automation in SPI context

Data Transmission on Mobile Networks

Test Automation for GPRS Testing

Preliminary Results and the way ahead

Test Automation Environment for GPRS Testing



HATT (Host Automation Test Tool)

The HATT performs tests in automatic way on simulated environment.

A simple language allows:

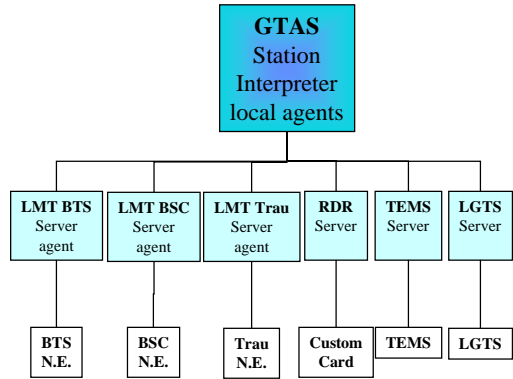
- to send, receive and trace messages among tasks;
- to perform the function calls and the function results manipulation;
- to give some simple control structure;
- to check and update global variable values;
- to interact with the object of the operating system, for example semaphores.

HATT automatically executes a sequence of test drivers listed in a file (*Test Chain File*). In every line of this file there is the name of another file (*Test Script File*) which contains a list of commands used for sending, receiving and checking messages, setting names of logfiles, accessing processor memory etc..

LSU Plus (Line Server Unit Plus)

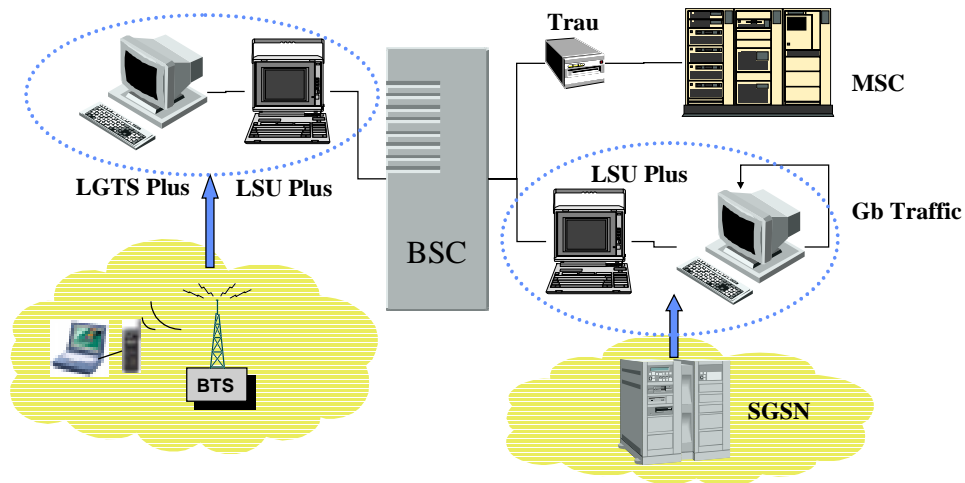
- Multiprocessor system with HW and SW developed ad hoc to satisfy the testing of complex telecommunications equipments such as the BSC in Circuit Switched and GPRS modality;
- handling of GSM lower layer protocols as support of Network element simulation: LAPD, MTP, TRAU FRAME, MAC/ RLC, Frame Relay, Network Service, BSSGP;
- Logging of signaling/ data channels;
- Special features for GSM traffic channels (TRAU frames, PCU frames, V.110, HSCSD);
- Static and dynamic configuration management;
- Router and remote access TCP/IP server;
- GPRS User data traffic generator;
- Satellite delay;
- etc..

GTAS (GSM Test Automation Tool)

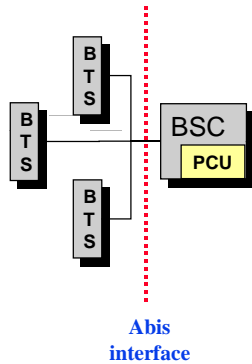


- System oriented for target testing based on a client/server architecture where the central station acts as a client interacting and controlling execution of a set of servers tools;
- Communication based on TCP/IP over LAN;
- GTAS station works by a interpreting test scripts that describe the actions to be executed on the remote devices and controls the results;
- Centralized execution and result checking is a significant step towards test automation.

A simulated test scenario



Abis Applications



LabLog

it is a monitor tool of the messages at the level of PCU frames.

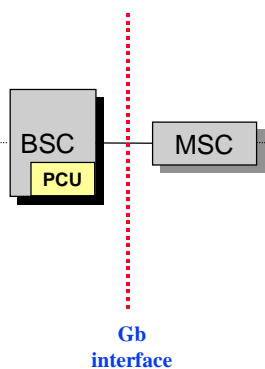
LabRes

it allows the sending and checking of received packets telephonic procedure messages driven by script files.

LGTS Plus

it allows to simulate a traffic scenario with both Circuit Switch (CS) and GPRS mobile stations.

Gb Applications



GbLog

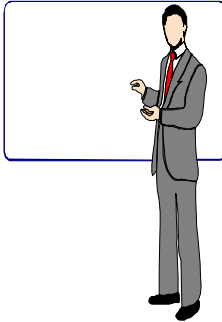
it is monitor tool of the messages at any level of protocol stack.

GbSeq

it allows the sending of messages interactively, driven by user or by file: the interface is permitted at any level (FR, NS, BSSGP).

Gb Traffic

it's a tool providing a loop path for the signaling and data traffic generated and monitored by the Abis simulator on MS couples.



Company Profile

Evolution of GSM and Mobile Telephony

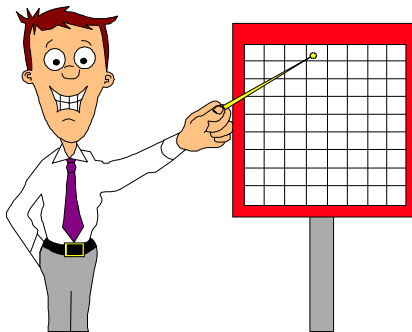
Software Development and Testing Processes

Test Automation in SPI context

Data Transmission on Mobile Networks

Test Automation for GPRS Testing

Preliminary Results and the way ahead

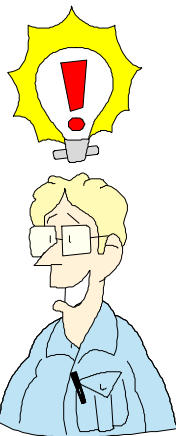
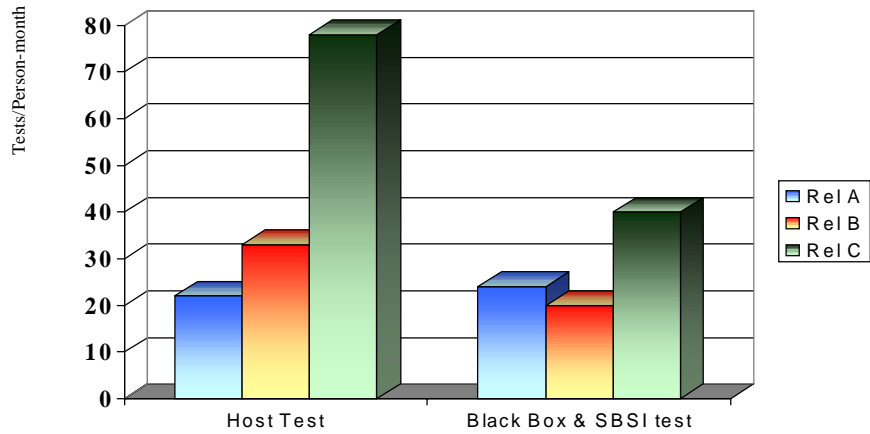


Preliminary results

In the areas where test automation has already been deployed:

- test productivity has improved;
- fault density has improved.

Test Automation Effectiveness analysis results



The way ahead

Our roadmap foresees at the time being:

- field fine-tuning of all the applications till now developed;
- full deployment of tools according to user requirements;
- deployment of various tools to other sections of the SBS projects.

Thank you



AUTOMATED TEST REUSE FOR PRODUCT FAMILIES

Mika Salmela, Jukka Korhonen, Jarmo Kalaoja

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN-90571, Oulu, Finland

Tel. +358 8 5512111, Fax. +358 8 5512320, E-mail: Mika.Salmela@ele.vtt.fi

Abstract This paper proposes a testing method applicable for validating configured software products. The configuring of test material in this particular method is based on software configuration. This kind of approach calls for a formal software production process, which uses product features or requirements. The method proposes links between product features and tests. The key issues of test configuration and management are discussed. These include test componentisation, test suite structuring, and tool support. The approach is demonstrated and evaluated by means of an example system.

Keywords Software testing, reuse of tests, configured systems, feature-based software

1 Introduction

New efficient software production techniques are important for improving the time-to-market of software products. One example of such advanced techniques is the so-called feature-based software production [Kalaoja et al. 97], which uses high-level requirements or features to help finding and selecting reusable software components for building a new product. The technique is based on a feature model representing the distinct features of a software product family, and also providing means for application configuration.

Though that kind of model-driven software development has shortened the production time, the validation of configured products still remains a bottleneck. A typical, straightforward solution is to use regression testing.

In general, regression testing is initiated by a new requirement, which is why the program and its documentation have been modified and need to be tested. The goal of regression testing is to convince the maintainer that the program still performs correctly with respect to its requirements. However, the term *regression testing* is just a common name for a re-testing process for modified software. It does not instruct us how to create reusable test cases, how to configure test suites from existing test material, nor does it say if it is reasonable at all to strive towards reusable tests. To begin regression testing, the test organisation and personnel involved need to decide on the outline of the test procedure, or create a new one [von Mayrhauser et al. 94]. In this matter, little help can be drawn from literature.

An effort to apply regression testing techniques to configured software products shows that they are not well suited to meeting the new testing challenges. Along with the feature-based development paradigm, the production of new software is rapid, and we do not want to slow down the process with any inefficient validation procedures. Somehow, the production of test suites, including the testing itself, has to be brought to the same level with the production of software.

To solve the problem, we could try to apply the idea of componentisation from feature-based production to testing. This implies that we would have a set of reusable *test* components, which could be configured to cover the characteristics of the product being tested.

This basic idea brings out many questions. If we decided to apply the idea, what kind of requirements would then be set on testing, testing process and test case design? What would the implications be for test automation and tool support? Could similar benefits be gained in feature-based testing as in feature-based software production? These, and other related testing issues, are discussed and evaluated in this paper.

2 FEATURE-BASED SOFTWARE PRODUCTION

The purpose of feature-based software development is to meet the stringent requirements of system production today, which has to be able to satisfy the needs of the customer in a rapid and profitable way. The list below defines the characteristics required from system production:

- time-based competition
- quick response
- fragmented markets
- proliferating variety
- increased customisation
- continual improvement
- shortening product life cycles
- cycle time reduction

Feature-controlled product configuration is targeted to addressing most of these requirements.

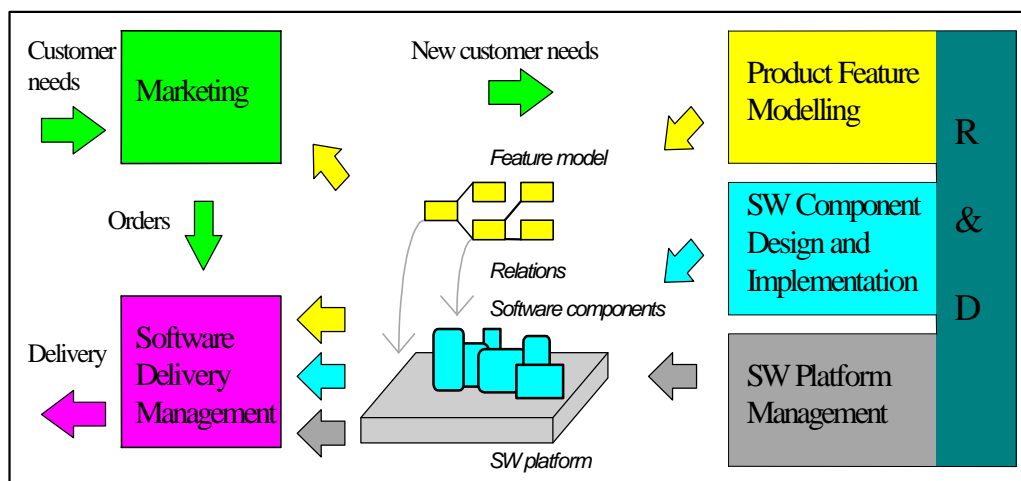


Figure 1. Feature controlled software development and delivery process [Kalaoja et al. 97].

The figure shows how the feature-based software production process has two main sections; software delivery and software development. Marketing and delivery management use the domain

models developed and maintained by software development. The essential model in the reuse and production of software is the feature model, which describes all available features for the product.

Marketing specifies the product that the customer wants by creating a *feature model occurrence*. The feature model provides assistance by presenting the existing features, which are re-used to the greatest possible extent. At best, the new product can be configured entirely by combining some of the individual features of previous deliveries. If this is the case, the software product can be assembled simply by integrating existing software components on top of a standard software platform.

The feature-based production process seems to offer an excellent platform for applying regression testing principles. The feature model defines the features for the domain, thus providing a solid basis against which system level testing can be conducted. Considering the goals of testing, one might propose an ad-hoc concept by asking why not create a testing model similar to the feature model, where test cases could be selected as features from a feature model. The applicability of this and other ideas are discussed in the following chapters.

3 FEATURE-BASED TESTING

The Feature-based testing approach proposes a testing method tailored to configured software products. The approach provides means for building tests and test environments efficiently. For the term *feature-based testing* we will use the following definition.

Definition 1. Feature-based testing is a method for testing product families. It uses test components and product feature descriptions for modifying and configuring tests.

Considering the previous definition, and the potential of feature-based development, the method should be able to answer the following questions:

- How to take the application domain knowledge into consideration in test design?
- How the features of the software product can be used for selecting proper test material?
- How the information of the feature model can be used to adapt tests to a desired software configuration?
- How to take mutually dependent features into consideration in the test suite structure and test implementation?
- How the identified test material is developed into executable tests?

3.1 Linking test material with the feature model

Linking test material with the feature model is easier if the material has been systematically arranged. Various criteria can be employed in organising the test material; e.g.,

- test type (e.g. these tests are intended for *system testing*);
- features or requirements linked to the test (e.g. this test is associated with the Display Refresh *feature*); or
- types of features or requirements linked to the test (e.g. these tests are intended for the *performance testing* of the Display Refresh feature).

Functional structuring can also be applied as a basis for the classification, meaning that the functional descriptions of product features are used for structuring the tests.

Since there are several ways of organising the test material, test methods and tools should have only few limitations as regards to defining the organisation and linking test data. As Figure 2 proposes, the user may find it necessary to link test plans with the feature model, or with the test environment configuration data. Free building of links should be made possible by the support system.

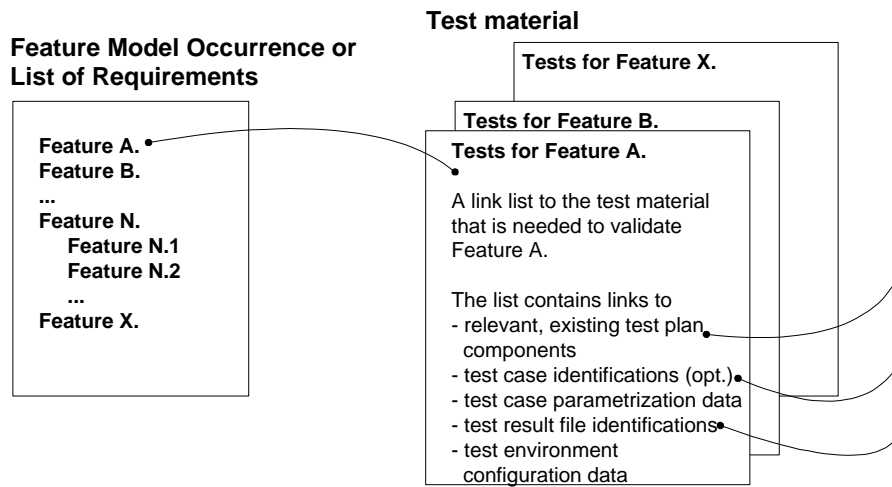


Figure 2. Linking the test material with the feature model or the requirements.

The features and other objects associated with the tests are subject to continual change. This leads to changes in the application itself, and therefore also those in test plans, test cases, scripts and data. This maintenance burden can be eased if the supporting system is capable of generating reports presenting the test material that may be affected by the changes. Updating links and adding new material should likewise be easy.

3.2 Processing the test material

The test material, now structured and linked with the feature model (or with other suitable objects, e.g. requirements), has to be developed further for generating executable tests. There are, in fact, several choices for processing the test material further, depending on the test execution environment. It may well be that there are no such test facilities available that could utilise executable tests. In this case, the support for the testers is a list of relevant tests and other test material. The testers will then *manually* execute the necessary tests.

If the processing of test material is taken a bit further the test supporting system could, for instance, provide modified test scripts and test environment configuration data for the testers. This could be called a *partly automated* solution.

The most advanced case involves a test environment capable of executing the tests. In this case, the test material has been *automatically* configured in such a manner that the features of the product are taken into account by the tests and that the tests can be executed without any additional manual processing.

3.2.1 Mutual dependencies

For automating the test execution, it is necessary to consider the implications that a feature model has concerning the test material. It may well be that features have, for instance, mutual dependencies, and therefore a solution in which a single *static* test suite is used for testing one feature is not reasonable. The feature model occurrence has to be used for selecting and modifying the test suites for the product.

Definition 2. Features are mutually dependent if the selection of one feature changes the behaviour of the other feature.

Let us assume that we have included feature *F1* along with some other features in a product configuration and the resulting product needs to be validated. Test scripts *S1... Sn* and corresponding result files *R1... Rn* have been developed for the domain. Domain analysis has revealed that feature *F1* changes its behaviour depending on the feature combination. In a test design, this would appear as follows:

- If *F1* is selected, then script *S1* is run for testing *F1*, and results defined by *R1* are expected.
- If both *F1* and *F2* are selected, then script *S1* is run for *F1*, but the expected results are now defined by *R2*.
- If both *F1* and *F3* are selected, then script *S2* is run for *F1*, and *R3* is expected.

The point is that the testing of a feature may require several alternative test scripts and test result files. The selection of appropriate tests depends on feature configuration, like *F2* and *F3* in the example above. The implementation of test material structure and test configuration has to be able to handle the conditions described above.

3.2.2 Test componentisation

Feature-based software production offers certain advantage to test development compared to other software models. Domain analysis in the software process is a must, in which you "carefully bound the domain being considered, consider the ways the system in the domain are alike (which suggests required characteristics) and the ways they differ (which suggests optional characteristics), organise an understanding of the relationships between various elements in the domain, and represent this understanding in a useful way" [Nilson et al. 94]. The next step is domain modelling, which

"provides a description of the problem space in the domain that is addressed by software" [Krut, Zalman 96]. The results of these tasks are applicable to test design as well, especially for identifying examples of reusable test components. The models define the user's point of view to the product, while in some cases the test components can be directly extracted from Use Case descriptions. In fact, a commonality analysis of Use Cases may have already been done during the definition of requirements. The analysis examines and identifies the common parts of Use Cases as use case steps. If this information is available, common parts are already known, and test componentisation will be simple and straightforward. If Use Case descriptions do not exist, it is reasonable to start by creating the descriptions and proceeding with the componentisation from that point on.

Figure 3 illustrates how test cases are extracted from a use case. For each action in the use case there is a corresponding test step. Common actions (e.g. related to hardware or software environment) can be hidden in preambles, postambles, or other test components. These actions are needed for bringing the system to a desired state, to initiate or to end the actual test. Specific, feature related actions are separated in their own test steps or components.

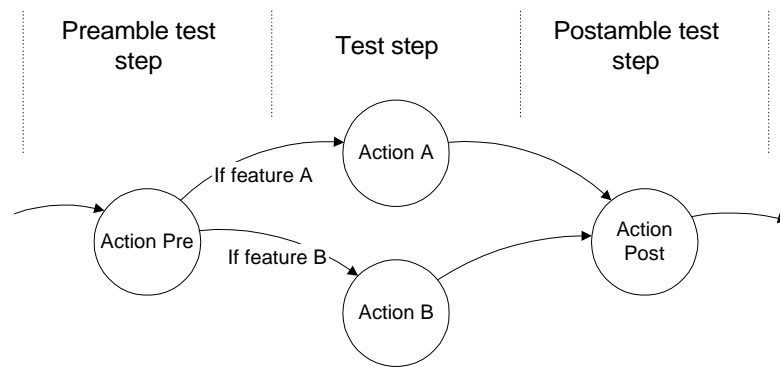


Figure 3. Abstracting test steps from a use case.

Use Case descriptions are good in modelling human user sequences. However, systems have various types of interfaces and they are seldom stimulated only by the human users. The communicating device could be a sensor or another system. In simple cases, the external device can be modelled using a static input-file, while devices with complicated behaviour need executable, dynamic models. These input-files and even system models can be handled like other test components; they can be parameterised and included in the test configuration when needed [Haapanen et al.97].

Test components can be structured as presented in Figure 4. The purpose is to hide the numerous test steps in test specifications and to make test planning easier. The specification (i.e. test controlling script) defines one Use Case step, but does not contain the actual test script, which is located in the component implementations. The alternative implementations are called by the specification component, which uses feature selections to determine the right implementation. Direct calls to implementations are not recommended, as they may lead to a loss of component reusability and thus increase the maintenance load. This solution is similar to the "information hiding" principle, which conceals unnecessary details from the user.

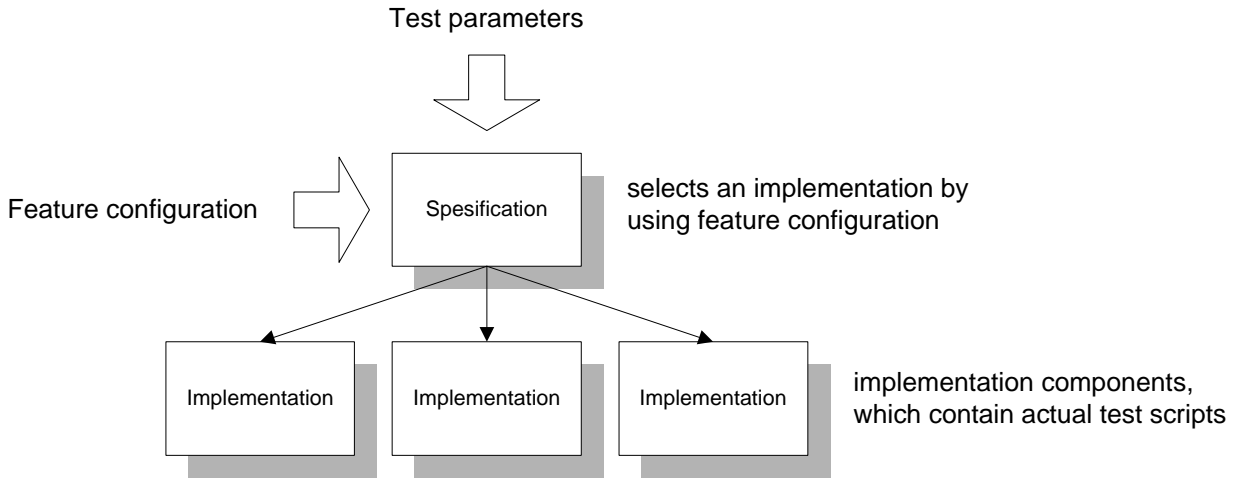


Figure 4. Test case componentation hierarchy.

The figure presents only one level in the test suite hierarchy. Test structuring can be continued as deep as needed, applying the principle of sub-scripting.

The selection of a test component implementation, i.e. test script, is defined in the specification using *if-then* clauses. The most general configuration is put into the last *if*-block, and the most infrequently used on top of the list. When the *if* -clause is used in this way, a new feature can be added into the system as a new speciality, while the old mappings/expressions remain the same. :

```

If (most specific feature configuration) then
  select script file 4
else if (specific feature configuration) then
  select script file 3
else if (common feature configuration) then
  select script file 2
else if (most common feature configuration) then
  select script file 1
else
  select default script file component
endif
  
```

Example 1. Selecting a proper test implementation.

3.2.3 Generating tests

The need for generating tests is related to test stopping criteria. If the system is simple, or the test set reduced, testing is typically stopped, when all the tests have been successfully executed. Sometimes static test sets are out of the question, for instance when there is a statistical requirement on test stopping. In this case test specifications can be enhanced to dynamic user models. Usage profiles, or other coverage criteria for implementing the dynamic control in test specifications can be applied.

Usage modelling can also be employed in test generation, by means of Markov chains or Finite State Machines, for instance. These approaches can be used for generating numerous stochastic tests, imitating the way a real user would use the system. Figure 5 shows an example of a Markov-type usage model. The graphic model can, if done by using a suitable description method, be compiled or executed as such. It is easy to notice that Markov models are very similar to use case descriptions, with the exception of the probabilities being included in the graphs. Thus use cases can easily be enhanced to Markov diagrams (Figure 3 and Figure 5). The model can be used as a top-level test controller. Probability conditions can also be manually coded in the test specification (see Example 2).

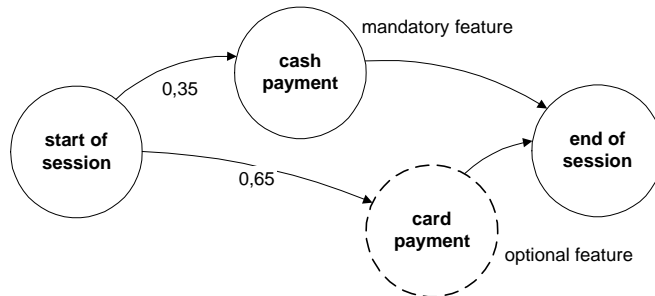


Figure 5. A usage model with probabilities.

The label *0,65* on the arrow means that the probability for the transition is 0,65 if the optional feature has been selected. The implementation of probabilities is easily accomplished by using a random generator with even distribution between [0, 1].

4 SUPPORT FOR A FEATURE-BASED TESTING APPROACH

Feature-based testing imposes specific requirements on supporting tools. At present, testing tools have few or no properties at all for supporting feature-based testing. Scripting language and component management are the key issues.

4.1 Script development

A script can be described as a sequential set of commands which mimics the normal controlling input to the application. Thus a script could be a reproduction of a keying sequence by the user for screen manipulation.

To implement hierarchical and modifiable test suites, a script language has to support

- logical condition constructions,
- data manipulation,
- external file input/output commands, and
- procedure calls with parameters.

As most script languages simply adapt to existing languages, e.g. C or Visual Basic, the requirements mentioned above are not hard to meet.

4.2 Test development and execution

Test development denotes the task of converting test scripts into executable tests. Test development has to support (note that we have the automated solution in mind)

- linking of test scripts to features; and
- sequenced running of test scripts from a test controlling script.

A test controlling script is usually necessary, e.g. for controlling the execution order of scripts. Some test execution tools support this requirement. However, commercial tools offer no support for implementing the first requirement.

4.3 Test storage and management

Test execution tools have usually no support for normal configuration management features. Thinking of real-life applications, this issue should be brought into the implementation. If no support for configuration management exists, a possible solution might be found in integrating the test tool with a configuration management system.

Efficient test storage and management calls for

- a version management system, which takes care of the test repository maintenance and linkage control;
- facilities for browsing test cases and searching test cases using different search methods; and
- capacity of modifying and rearranging retrieved tests using a suite manager.

5 A CASE STUDY

The case study shortly presents how the testing approaches mentioned above were put into practice. The demonstration system is based on two real embedded systems. The system employs several features, which have been linked with the test suite repository. KataSystems is an imaginary betting slot machine, consisting of a keyboard, a display and some external interfaces for various purposes, e.g. for checking the validity of a credit card. The system resembles teller machines. The KataSystems demo environment (Figure 6) consists of three components: the betting slot machine itself, a tester and a configurator. The configurator is used for modifying the outlook and behaviour of slot machine products. When the user configures a slot machine, the tester is loaded with tests needed to validate the selected features. The tester is used for executing and monitoring the selected test.

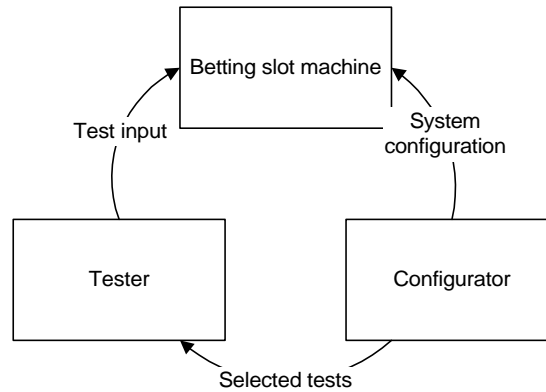


Figure 6. KataSystems demo environment.

Variations in the slot machine are modelled by using a feature model. For instance, the slot machine automatically accepts cash payment, with an option of credit card payment. The credit card payment has an additional option of payment correction, i.e. the bet can be changed. In addition there are options on display types and language.

The feature model is converted to a feature selector, which is implemented using check boxes and radio buttons. After the user has configured the system, the tester is loaded with tests corresponding to selected features. The tester chooses only those tests needed for testing the configured product.

Figure 7 presents the layout of the configurator. Mutual dependencies are shown as arrows referring to other features. The user or system configurator can create different kinds of systems by clicking the desired features, and then pressing the OK button.

Figure 7. Feature configurator for the demo system.

The selected features are imported into the test script as an *include*-statement. The control of testing correct features is gained by using the data of the *feature_model.h* file in conditional statements. Test data is stored in files, where it is read by using special routines. Note the implementation of a usage profile using a random generator (see also Figure 5).

```

/* The feature configuration of the application */
#include "feature_model.h"

/* Test parameters */
get_test_params( PAYMENT_TESTING, number_of_tests, *bet_sums );

/* Test execution */
for ( i=0; i < number_of_tests; i++)
{
    /* Preamble */
    start_user_session;
    enter_bet( *bet_sum, i );

    /* Feature configuration controls test selection */
    /* If both features included, apply usage profile */
    if ( CARD_PAYMENT && (uniform_distribution(0,1) <= 0,65 ))
        { card_payment; }
    else
        { cash_payment; }
    /* Postamble */
    enter_keycode( STOP_PAYMENT_SESSION );
}
}

```

Example 2. A part of the main test controller for the *bet placing* feature.

In the demo system we used the C programming language as a test scripting language. The C language has the capacity of creating reusable test components or functions. It also allows passing parameters to test components, which was one of the main requirements for the script language.

| Feature | Test suite |
|---------------------------|-------------------------|
| Payment | payment_test |
| Cash | cash_pay_test |
| Correction | payment_correction_test |
| DisplayTube | display_test |
| AG_Flat | ag_flat_test |
| feature list continues... | |

Table 1. Test suite correspondence to features.

The test suite hierarchy of the demo follows the structure of the feature model. On the left side of Table 1 there is a list containing all the features and on the right the corresponding test suites.

A simple way to implement links between test material and feature model is to use a test index file. The index file maintains links from features / functions / issues to test files, e.g. test specifications and scripts. When the user makes selections on the feature selector, necessary tests are searched in

the test index file. The test index file represents 'a small database' maintaining links between features and tests.

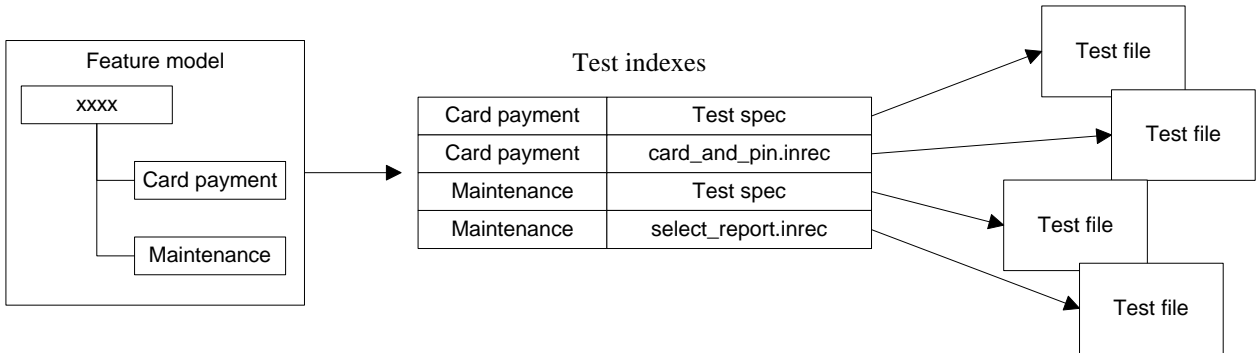


Figure 8. Test index file.

6 DISCUSSION

Reuse has generally been connected with software development and production, and it has relied on software componentisation. In software testing, reuse seems to be more of an ad hoc nature, inspired by practical considerations. More formal reuse procedures can be found in telecommunications, where standards give generic advice for reusable test design.


The feature-based testing method is mainly aimed at systems that apply features for defining their properties. This approach lines out the essential characteristics of test reuse. Among these issues are, for instance, test suite structuring, test script design and test material management. Some of the ideas have been presented earlier, though perhaps not implemented and evaluated. This paper integrates relevant ideas and examines the requirements for a test environment to support the feature-based approach.

The key issue of the approach is the structuring of tests into test components. The components have to be configured in a manner that corresponds to the features of the product. A test domain analysis, which can clearly make use of the feature domain analysis, has to be carried out prior to test structuring. The analysis identifies the test components.

Empirical evaluation of the results is essential, especially in this approach, which has a strong practical orientation. For this purpose, a demonstration system was constructed, so as to implement the substantial features and ideas presented in this paper. It was also equally important to evaluate the functionality and relevance of the proposed techniques and solutions. The designing of tests for the demonstration system was straightforward and uncomplicated and the configuration of tests revealed no major problems in the approach. The next step is to select a real software product family for testing the ideas generated through the process.

References

- ETR 141. Methods for testing and specification (MTS) Protocol and profile conformance testing Specifications: The Tree and Tabular Combined Notation (TTCN) style guide. 1994.
- EWOS/TA. Methods for testing and specification (MTS) partial & multi-part abstract test suites (ATS), rules for the context-dependent reuse of ATSS. EWSO/ETG 057. 1995.
- Haapanen, P., Pulkkinen, U., Korhonen, J. Usage models in reliability assessment of software-based systems. Special report. STUK-YTO-TR. 1997.
- Jeon, T., von Mayrhauser, A. A knowledge-based approach to regression testing. First Asia-Pacific Software Engineering Conference. IEEE Comput. Soc. Press, 1994.
- Kalaoja, J., Toivanen, J., Okkonen, A., Niemelä, E., Ihme, T. Configurable feature-based application software. KOMPPI-project report. VTT Electronics. 1997.
- Krut, R., Zalman, N. Domain analysis workshop report for the automated prompt and response system domain. Special report. CMU/SEI-96-SR-001. May 1996.
- Nilson, R., Kotgut, P., Jackelen, G. Component provider's and tool developer's handbook central archive for reusable defence software (CARDS). (STARS-VC-B017/001/00). Reston, VA: Unisys Corporation. 1994.
- von Mayrhauser, A., Mraz, R., Walls, J., Ocken, P. Domain based testing: Increasing test case reuse. Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processors. 1994.



Automated test reuse for product families

Mika Salmela
Jukka Korhonen
Jarmo Kalaoja





Objectives

- Develop testing method applicable for validating configured software products in product families.
- Apply the idea of componentisation from feature-based production to testing in order to improve reusability and configuration of tests.
- Establish requirements for testing tools.



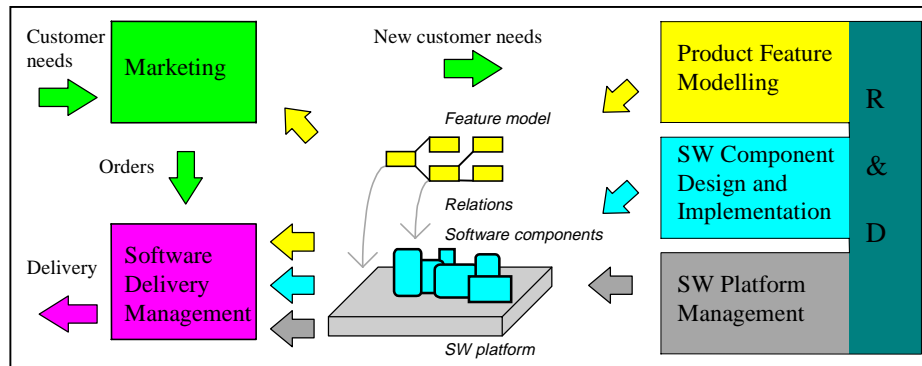
Feature based testing

- How to take the application domain knowledge into consideration in test design?
- How the information of the feature model can be used to adapt tests to a desired software configuration?
- How the features of the software product can be used for selecting proper test material?
- How to take mutually dependent features into consideration in the test suite structure and test implementation?

Feature based testing

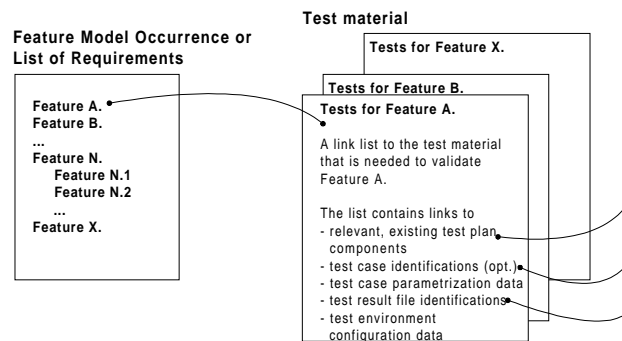
- Feature is any distinctive or unusual aspect in a system. Features are differentiates products in the product family.
- Feature-based testing is a method for testing product families. It uses test components and product feature descriptions for modifying and configuring tests.

Feature based software production



Linking test material with feature model

- Test material should be systematically arranged
- Items in feature model are linked with tests and further with other test material
- Free building of links should be made possible by the support system

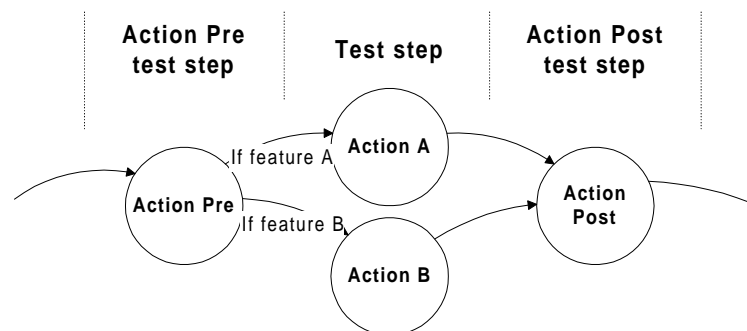


Linking test material with feature model

- Test material processing can be
 - manual
 - partly automated
 - fully automatic
- Features are mutually dependant if the selection of one feature changes the behaviour of the other feature.

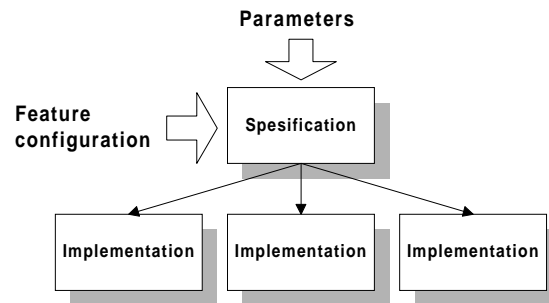
Componenting use case to test steps

- Use case description are used to identify instances of reusable test components
- For each action in the use case there is a correspondiv test step
- Feature variation points are handled with test componentitiation



Test case componentation

- One specification for each action
- Selection mechanism is defined in specification component
- The test designer calls specification components as reusable components
- Hides numerous implementation options



Script and tool requirements

- Logical condition constructions
- Data manipulation
- External file input/output commands
- Procedure calls with parameters
- Ability to define links between objects
- Version and repository management for test objects

Automatic Usability Testing for Hypermedia Navigational Graphs

Martín González Rodríguez

Multiple navigational graphs can be obtained as the result of the *design stage* of a hypermedia-based artifact. The only way to know which one adapts better to the user navigational metaphor is by means of usability testing. This technique is expensive in terms of the number of human resources needed to perform it, and it isn't able to record spontaneous user behavior. It also introduces external noise because users could feel nervousness or confusion while they are under observation.

The use of automatic testing tools is an interesting and cheap alternative that avoids the problems commented. We designed and developed our own automatic navigability testing system (ANTS), which can be used with any kind of hypermedia device including web sites.

I. THE HYPERMEDIA DEVELOPMENT PROCESS

Hypertext documents are one of the most popular communication channels nowadays. Thanks to the World Wide Web, this kind of document is used by millions of people all around the globe. Due to its popularity, the working of hypertext information systems is well known even by novices. The concepts of node, anchor and link are quite good implanted in the computing community. As user training is almost not required, hypertext documents can be cheaply introduced in educational environments (such as high schools and universities) where their use has been highly increased in the last years. In the mentioned educational environment, hypertext is a powerful and flexible source of information that is employed for both reference and knowledge transmission.

The arrival of multimedia improved the way hypertext worked providing it new data such as sound, video, virtual reality and so on, which help to teach concepts which were almost impossible to explain using only text. We are talking now of hypermedia. But this wonderful teaching tool only seems to work when the information provided by its nodes is well structured (see Nielsen 1993, *Usability and Navigation*, 125-143).

A. The Analysis Stage

The first task to do when building a teaching tool based on hypermedia is to clearly define the subject to be described and the academic goals to be reached. Designers must know from the just beginning what is going to be taught and what is the minimum amount of information the pupils must be learn in order to reach the expected educational success. This task is commonly known by some methodologies as the *analysis stage*.

Once the knowledge to be explained by the hypertext artifact has been selected and its goals have been completely defined, the first task to be done in order to build a high quality product is to perform a deep research of the subject matter. This will provide useful information about the knowledge to be explained and its structure. During this process, all relevant information must be compiled and organized into well-structured categories.

B. The Design Stage

The following task –commonly called *design stage* by some designing methods– is the most delicate in order to get a hypermedia product fully compliant with the idea of a cognitive artifact, that is, an artifact designed for the nature of human cognition. Applying more *humanness* to the design can brand the difference between a successfully or a unsuccessfully product in terms of educational and cognitive goals, so it must be done quite carefully.

The main objective of this task is to structure the whole knowledge to be transmitted into smaller didactic units, following a divide and conquer strategy. Every didactical unit will represent a conceptual node in the resulting hypermedia system.

At the beginning of the *design stage*, the knowledge base fits in only one didactic unit, which must be divided by designers into different and simpler units. Every one of these units can be divided too, structuring the resulting knowledge in a linear narrative fashion (figure 1), a hierarchical one (figure 2) or a combination of both.

This is a recursive process that will provide the full list of nodes of the hypermedia artifact as result. Every node will represent the smallest knowledge unit possible and will be

considered as an autonomous source of information (see Nielsen 1993, *Nodes and Links*, 101-106). Once the user arrives to one these nodes in the final product, he or she should be able to obtain full information about the subject represented by it, knowing that the information will not be replicated or complemented by any other node inside the scope of the hypertext artifact.

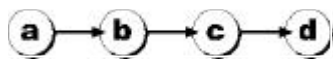


Figure 1

The resulting nodes are linked each other by hyperlinks, creating a directed graph structure generally called *navigation model*, using the terminology of some hypermedia design methods such as the Object Oriented Hypermedia Design Method (OOHDM) (see Schwabe and Rossi 95). The importance of this structure becomes obvious. It will determine how the user moves around the nodes and –even more important– how he or she retrieves knowledge from them and from the whole hypermedia product. It is clear that an adequate design of this graph structure is fundamental in order to reach the educational goals previously defined.

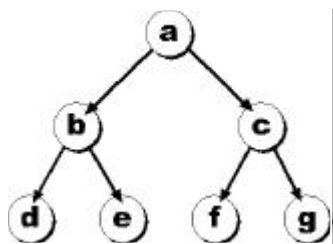


Figure 2

The *design stage* is not only the most important of the whole hypermedia development process but also the most difficult to perform. It is common to end this task with multiple resulting graphs depending on the different approaches used to explain the knowledge to be transmitted (Gonzalez 1999). For example, in a hypermedia tool for teaching history we could try to explain the subject at least from a temporal or from an entity point of view. If we follow the first approach, after an initial designing process, we might divide the history knowledge into historical ages, such as *Prehistory*, *Ancient Age*, *Medieval Age* and so on. On the contrary, if we follow the second approach, we could end this process with nodes labeled such as *The Roman Empire* or *The World War II*. This is the approach used by Yahoo, the popular internet hypertext based searcher (<http://www.yahoo.com>).

The complexity level increases even more if we consider that the resulting graphs may depend not only of the knowledge to be transmitted but also of the kind of user who will receive the information (see Olsina 1998). In the mentioned example of a hypermedia kiosk for a history museum, it is clear that designers don't affront the same

problem when designing the hypermedia model for teaching history to children that when they do it to teach history to adults. In this example, the age of the user represents the category that determines the kind of user, but in other cases it could be any other, such as for example the experience of the user. In this later situation, it would not be the same to teach history to the casual museum visitor that to teach it to an archaeologist. The expectation and interests of both kinds of user are quite different.

The nature of certain kind of hypermedia products such as hypermedia kiosks or web sites, implies the necessity of designing them to comply with the expectation of more than one kind of user (see Olsina 1997). For example, the navigational graph of a hypermedia kiosk intended to be used in a museum, should be easy enough to introduce novice visitors into the exhibition theme and at the same time, it should be complex enough to satisfy the curiosity of expert visitors.

In many cases both goals are so opposed that they can not be satisfied by a single navigational graph, so more than one is required. In this case, some kind of cognitive analysis must be performed on the input given by users in order to promote them to another navigational graph (see González 1999). This promotion can be done by a single direct test or by observing the kind of information the user is retrieving from the hypertext system. Some hypermedia design methods such as OOHDM allows the design of different graphs from a unique knowledge-base (see Schwabe 1995). This goal is reached by the design of different navigational graphs depending on the different kind of final user.

Figure 3 shows how the knowledge base identified during the *analysis stage* could be splitted in two (or more) different nodes during the *design stage*. Although the knowledge base is the same for both nodes (novice and expert), the amount of information provided by every one isn't the same. While novice users may be happy with general concepts, expert users normally require a high amount of detailed information. However, this isn't a static model as users are able to change their status by mean of evaluation. Intelligent hypermedia systems should be able to track the level of knowledge of every single user in order to promote him or her into a higher navigational graph.

Even when a simple graph can be used to satisfy the knowledge necessities of more than one kind of user, the contents of every single node can be designed with different approaches depending on the target user. The different designs for the same node are generally related with the language used (formal, informal, for adults, for children, etc.) and cognitive factors related with universal access to the interface for people who have a physical disability that requires additional access methods. People in this group of users mainly have difficulty with computer input devices such as the mouse or keyboard so a proper user interface must be provided to the final view of this kind of nodes (see Apple 1992).

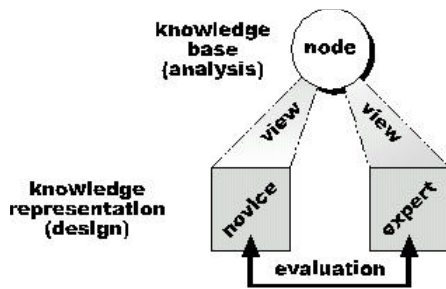


Figure 3

Another important application of this multiple view of a single node is the localization of its contents to different cultural values. In this case, every single node will have as many views as many different cultural values the hypertext system must support.

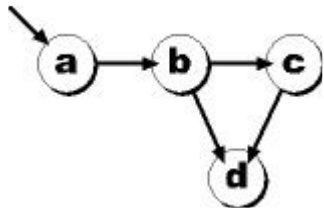


Figure 4

Figures 4 and 5 show possible navigational graphs for the same hypermedia application. Navigation on graph 1 start with node A and finishes with node D. In the example, users intended to use graph 2 don't need the information contained in node A; however they might need an additional node (E), not present in graph 1. The contents of a single node for both navigational graphs shouldn't be the same, so node B in graph 1 may not have the same content as node B in graph 2. Entry nodes for the navigational graphs neither are the same (Node A for graph 1 and node B for graph 2).

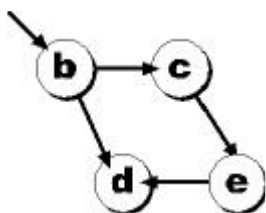


Figure 5

C. The Testing Stage

Once the *design stage* has finished, it is time to convert the navigational graphs into a fully working educational tool such as a web site, a hypermedia application or a multimedia kiosk. The *developing stage* is the most expensive one, so it is necessary to assure the quality of the navigational graphs before translating them to code.

Unfortunately, during the *design stage* there is not a clearly way of knowing whether the behavior of the final user will match what it was expected. Once the different navigational graphs have been designed, how can designers decide which of their designs will match user requirements?

The best way to measure how useful a navigational graph will be to the target audience is by exposing it to the scrutiny of users. This exposure can be done by prototyping. Under this approach, the developing of preliminary versions of the navigational graph is required to verify its workability (see Apple 1992, 41). Once the prototype is ready, it can be verified by usability testing, watching and listening carefully to users as they work with the prototype when they are encouraged to perform certain tasks.

The main goal to be reached during the *testing stage* is to get as much information as possible about how the final user moves around the set of different navigational graphs proposed. The information required consists usually of the following items:

- The list of the most visited nodes.
- When these nodes are visited.
- Time of user arrival/departure to/from a node.
- How long user remains in a single node before moving to another one (how long user takes to obtain the information he or she was looking for)
- Which are the most popular destinations (links) from a single node.
- The set of visited nodes during a single user session (including its order).

All data collected provide quite valuable information in order to determine the cognitive model of the user and allows designers to make accurate predictions of the user navigability behavior. This information should be contrasted with what designers expected in order to measure the quality of the design.

Usability testing is a really useful technique in order to improve the behavior of the user interface of many devices and software applications. It also was revealed as a powerful technique for revealing hidden user behavior, and for discovering internal user-generated metaphors when dealing with the user interface of some machines (see Shulz, Van Alphen and Rasnake 1997). However its use to perform validations on navigational graphs has some

disadvantages which can make it a slow and expensive process.

Classic usability testing requires permanent observation of the user. As the information to be obtained requires high precision (time of arrival and departure, time expended on a single node, navigational path, etc.) each user needs at least one observer. As navigation thorough the graph is not a simple task, long time of observation will be required.

Another important disadvantage of using usability testing for this kind of validation is that users who participate in the experiment know that they are being observed. This situation may add external factors to the testing such as nervousness, confusion, timidity and son on, which could influence user behavior when navigating through the hypermedia artifact. The sole presence of a camera (sometimes needed to record user behavior) modifies perceptively people's attitude, so it needs to be hidden.

When testing user interface, users need some kind of training such as explaining them how to think loud during observation, saying what comes to mind as they work. Although thinking loud is not extremely necessary when evaluating navigational graphs, some kind of training is required too.

We mustn't forget that users are dedicating an important part of their free time to our experiment, **so it must show a** clear purpose or users could think that they are working for nothing. Observers must inform users about the purpose of the experiment and should give them a set of tasks to be performed with the prototype. The knowledge about the experiment can influence navigability, because user can know or at least guess, what is expected from them, adapting their own internal navigational metaphors to the navigational model of the prototype provided.

Both factors (being observed and knowledge about the experiment) have the negative effect of destroying part of the spontaneous user behavior, which is just what the designers are looking for when testing navigational graphs.

Another interesting handicap of human conducted usability testing is that it is quite difficult to discover new kind of – not already identified– users, as participants selected for testing have to match the same demographic background and experience level as typical users in target audience. It might happen that a completely different audience could use the final product, as it is the common case of hypermedia kiosks.

Summarizing, although usability testing is a powerful tool for evaluating user interface and user behavior in hypermedia environments, it is a mechanism too expensive and slow to be applied to the validation of navigational graphs where quite special data must be obtained quickly. This tool can also modify user behavior, obtaining wrong data values as result.

II. ANTS: AN AUTOMATIC NAVIGABILITY TESTING SYSTEM

If designers want to avoid subjective influence during usability testing, they should never bring volunteers to the laboratory. The ideal condition to validate prototypes will be to test them under the same conditions that the final product will be used, that is, testing it in the user-computing environment. When testing navigational graphs under this approach, users should never know anything about the experiment, even the role they are playing. Observers must change their behavior too, as they can not be present during user navigation due to obvious reasons.

Under this ideal situation, users feel free to explore the navigational graphs provided, so external factors will not be added to the results obtained. Obviously, experimentation using this technique needs a completely new approach and additional tools to support usability testing in situations not under observer's control.

In order to reach this goal, we have developed an automatic navigability-testing tool, which we have called **ANTS (Automatic Navigability Testing System)**. The aim of this tool is to observe users while they navigate through any kind of hypermedia artifacts, registering their navigability behavior on a database for a post hoc analysis.

As the current implementation of ANTS was coded using Java technology, it can run in almost every computing platform, including Macintosh, Unix or Windows. Java is also supported by a healthy collection of Internet browsers (such as Microsoft Explorer or Netscape Communicator), so ANTS can be used for testing navigational graphs of both, platform dependent hypermedia applications and web sites on Internet.

Another quite important advantage of using Java for the implementation of this kind of systems, is the great versatility of its communication libraries and the variety of communication frameworks provided, such as sockets, RMI or CORBA. Also, the servlets technology included with the Java 1.2 release might allow ANTS to test user navigability through nodes dynamically created in the domain of adaptive hypermedia.

A. *The Anthill Metaphor*

The design of ANTS is based on the client-server paradigm, following a metaphor based on the life of a community of ants. In this metaphor, the ants (very small agents) go out the anthill (a central server) looking for food (the information) in every picnic available (a single user navigability session). Once the ant gets its food, it comes back to the anthill and stores it carefully in one of the anthill's warehouses.

When testing, the ants that serve as agents must be included **in every node of every navigational graph**. This task can be performed automatically by some Hypermedia developing tools based on Java (such as the European Project JEDI, see Transtools 1998). Once the ants have been included, the testing version of the product is ready to be distributed among the volunteers who participate in the usability test.

Once the user has arrived to a single node during a navigation session, the ant inside the node is downloaded from the anthill (the server). When the ant arrives to the user machine, it establishes a connection with the server, which is used to send the data collected. In the current version of this system, the ant sends a complete identification of the node visited, along with the specific instant of time of arrival.

If the security manager of the Java Virtual Machine allows it, the ant might collect information about the user too (see Weber 1997). User identification depends on the kind of prototype being tested. For web sites, a combination of the Internet Protocol Address (IP) and the host name of the user machine may be enough. For hypermedia applications, the agents could complement this identification with the user account ID, the product license number or even better, with a specific code provided to the user by the designers.

Our ants also report other events, such as for example, the time when the user abandons a single node. This task is performed even when the user moves to a different application, a common practice detected in multitask environments.

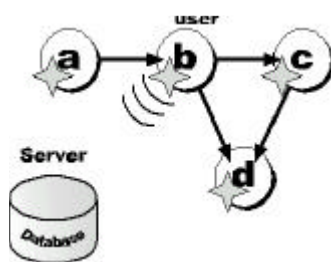


Figure 6

Figure 6 show an example of a navigational graph monitored by ANTS. Every single node has its own ant (the gray star) but the only one who broadcast information to the server is the ant owned by node B, as it is the node who is being visited by the user.

Future extensions of the ant agent class would allow the creation of even more elaborated reports. They could include the set of components selected by users, the list of the window locations clicked by them or the collection of components that are visible or hidden during scroll tasks. The kind of information provided by the little ants will depend on the information required by designers in order to improve the quality of their products.

B. Configuring an Ant

The information received by the anthill (the server) is compiled and organized depending on the navigational graph that is being tested. The multi-threaded nature of the server's design allows the concurrent testing of more than one navigational graph. In fact, there are no limits to the number of multimedia projects that can be tested concurrently by ANTS.

In order to know where to store the information received from an ant, this little agent must notify the server about the project that it is monitoring. To perform this task, an identification code must be assigned by designers to every navigational graph under testing. The parameters needed to setup a right configuration for an ant agent include that identification code and also another identification code for the node the single ant is controlling. Both codes must be unique and can be easily assigned by a hypermedia development tool.

The piece of html code below shows how easy an ant agent can be configured for testing a web site. Parameter SITE indicates the code assigned to the navigational graph (in this case a web site called *EDIWeb*) while parameter ID shows the code assigned to the node (the web page *EDIEval/c-intro.html* inside the *EDIWeb* site), which gives a clearly indication of the node visited by the user. Other parameters are implementation dependent and indicate the socket connection port and the host where the server is running.

```
<applet
code="uniovi.cognition.lure.Ant10APP.class"
codebase = "../ServWeb/BinJava"
width=10
height=10>
<param name="HOST" value="polar.uniovi.es">
<param name="PORT" value="1932">
<param name="SITE" value="EDIWeb">
<param name="ID" value="EDIEval/c-intro.html">
</applet>
```

The Java code below shows the equivalent ant configuration for the node's constructor of a desktop-based hypermedia application. Identification for the node and its navigational graph must be provided as part of the constructor's parameters.

```
/**
a constructor for the node

@param theGraphID the graph identification
@param theNodeID an identification for the node
*/

public node (String theGraphID, String theNodeID)
{
// setup for the ant agent
ant = new Ant (polar.uniovi.es, 1932, theGraphID,
theNodeID);

// Other actions to be performed by the constructor
...
}
```


C. Testing Navigability using ANTS

Once the designers have finished the prototype to be tested, it can be distributed among the volunteers, as it would be done with the beta version of the final product. For those users without Internet access a version of the server must be provided in order to capture the data generated by the usability test. In this case, the server should start when the users run the prototype and stop when they quit the application. If the hypermedia artifact to be tested is a web site, the only action to be done is to install it in a host acting as web server. If the usability testing is going to be performed on a multimedia kiosk, the prototype should be installed under the same conditions under the final product will run.

Because users don't know that they are being observed, users will work with the product, as they would do under normal circumstances. Using this approach, navigation through the hypertext system will not be altered by the external factors mentioned before such, as nervousness or confusion. Only when the experiment finishes, volunteers will be informed about the technique used.

Notice that volunteers are informed, from the just beginning of the experiment, that they are participating in a usability testing. The only fact they ignore is the observation technique used by researchers. This is a common practice in merchandising, where researchers observe customer behaviour through hidden video cameras (Masson and Wellhoff, 1990).

This approach not only eradicates irrelevant noise from the experimentation process but makes it also cheaper a more efficient too. By mean of automatic testing tools, the data collected flows freely from the source to the server storage system, where it can be analyzed off-line. Testers are liberated from the boring task of capturing data, so they can focus their efforts in analyzing the results obtained, in order to improve the navigation of their hypermedia designs. As there is no need to assign human resources for conducting user observation, the whole process of usability testing is cheaper. This way, many more volunteers can be observed using limited human resources.

The following report was obtained using ANTS when testing user behavior for the navigational graph of a web site. User identification corresponds to the IP Address (159.55.11.111) and the host name of the machine (pinon.uniovi.es) used by the user to navigate to the web site. The arrival was at 17:50:45 Hrs. (server local time) from a computer located inside the scope of the ECT time zone. User movement along the web site was also registered. Time expended in every single node and in the whole site is also computed from this data.

```
DATE: January 16th 1999 (Saturday)
TIME: 17H50' GMT+00:00

IP ADDRESS: 195.55.11.111 (pinon.uniovi.es)
TIME ZONE: ECT (-2)
```

```
LOGS: (All dates are local to server)

17:50:45 -> Log in.
17:50:46 -> User arrives to site.
17:51:08 -> User moved to <Computing/InXena>.
17:52:01 -> User abandons site.
17:52:12 -> User moved to <Agenda>.
17:53:23 -> User leaves site definitively.
17:53:23 -> Log out.

User was here about 00:02:35 seconds.
```

When compiling data about every navigational session in a site, important information can be obtained, such as which are the most visited nodes, which are the links more suitable to be chosen by the average user, and so on. With all this information, testers are able to contrast the average time spent by user in each node, with the expected values calculated during the *design stage*. This information is quite useful in order to know whether the design of the navigational graph is realistic or not.

The accurate, reliable and quickly collected information provided by automatic usability testing tools allow this kind of researches, monitoring real user behavior in the fastest and cheapest possible way.

III. ACKNOWLEDGEMENTS

Thanks are especially due also Agueda Vidau Navarro. ANTS-based technologies are supported by the Oviedo3 project.

IV. REFERENCES

- [1] Apple Computer Inc; (1992): **Macintosh Human Interface Guidelines**. Adison-Wesley Publishing Company. ISBN 0-201-62216-562216
- [2] González Rodríguez, Martin; (1999) **A Hypermedia Development Process for Jedi-Leia**. Cuadernos de Investigación Ingeniería Informática (2). Editorial Servitec, Oviedo.
- [3] Masson, J. E.; Wellhoff, A.; (1990) **El Merchandising: rentabilidad y gestión del punto de venta**. Deusto, Madrid. ISBN 8423405117.
- [4] Nielsen, Jakob; (1993) **Hypertext and Hypermedia**. Academic Press, Cambridge. ISBN 0-12-5118410-7.
- [5] Olsina, Luis Antonio; (1997) **Process Model in the Hypermedia Development**. (1997) *Conferences at EUITIO, University of Oviedo*. <http://www15.uniovi.es>
- [6] Olsina, Luis Antonio; (1998) **Cognitive criteria in the development of Hypermedia Applications** *Conferences at EUITIO, University of Oviedo*. <http://www15.uniovi.es>
- [7] Schwabe, Daniel; (1995) **Summary of OOHDM**. <http://www.cs.tufts.edu/~isabel/schwabe/fig1.html>
- [8] Schwabe, Daniel; Rossi, Gustavo; (1995) **Abstraction, Composition and Lay-Out Definition Mechanism in OOHDM**. *Proceedings of the ACM Workshop on Effective Abstractions in Multimedia*. San Francisco, California. <http://www.cs.tufts.edu/~isabel/schwabe/MainPage.html>
- [9] Shulz, Erin; Van Alphen Maarten, Rasnake William (1997); **Discovering user-generated metaphors through usability Testing**. *Proceedings of the Second International Conference on Cognitive Technology*. Aizu, Japan.
- [10] Transtools; (1998) **JEDI: Java Enabled Database over Internet**. ESPRIT Project (EP24231). <http://www.transtools.com/jedi>.
- [11] Weber, Joseph; (1997) **Special Edition Using Java 1.1**; Third Edition. ISBN 0-7897-1094-3

Automatic Usability Testing for Hypermedia Navigation Maps

Martín González Rodríguez
University of Oviedo

Developing Hypermedia (HDP)

- I- Analysis Stage.
- II- Design Stage.
- III- Testing Stage.
- IV- Developing Stage.



Testing Stage (1)

- There is not a clear way to predict how users will navigate along the hypermedia device.
- There are too many cognitive factors involved.
- How can designers decide which Navigation Map adapts better to user behaviour?

For more information...

Time and Web <http://www.soc.staffs.ac.uk/seminars/web97/>



Testing Stage (2)

- A possible answer is by **Prototyping**.
- Once a prototype is ready, it can be verified by **Usability Testing**.
- **Goal:** To get information about how users move around the set of Navigation maps provided.
- Information should be contrasted with what was expected during the design stage in order to measure design's quality.



Testing Stage (3)

- Information required include –but it is not limited to– the following items:
 - The list of the most visited nodes.
 - When these nodes are visited.
 - Time of user arrival/departure to/from a node.
 - How long users remain in a single node.
 - Which are the most popular destinations (links) from a single node.
 - The set of visited nodes during a single user session.




Usability Testing Advantages

- Quite useful technique to improve the user interface of many devices and software applications.
- It can be used for revealing hidden user behaviour when dealing with the user interface of some machines.
- Usability testing revealed as a powerful technique for discovering internal user-generated metaphors.




Usability Testing Disadvantages

(1)

- Requires permanent observation.
 - Each volunteer needs at least one observer.
 - It is an expensive and slow process.
 - Prototyping requires quick evaluation in order to improve prototype's quality.
 - Testing quality diminishes as samples are generally small due to economical reasons.
- 

Usability Testing Disadvantages

(2)

- It is quite difficult to discover a new kind of –not already identified– users.
 - Volunteers know they are under observation:
 - Add external factors to the navigability testing.
 - Knowledge about the experiment might influence navigability.
 - Destruction of spontaneous behaviour (user own navigational metaphor).
- 

Ideal Situation for Testing

- Volunteers should never know the role they are playing during the experimental process.
- Designers should never bring volunteers to the laboratory.
- Usability testing restricted to the same conditions under which, the final product will be used (**Remote Testing**).

Advantages of the Ideal Situation

- User will feel free to explore the Navigational Maps provided.
- Testing is not affected by External Factors.
- Simplifies the process of detecting new navigational metaphors.
- Information about how the user-computing environment affects navigation can also be observed.

Remote Testing Techniques

- As observers have no direct contact with the volunteers, testing under this Ideal Situation will require:
 - A new technique of experimentation where users aren't asked to perform predefined tasks.
 - Tools for supporting remote Usability Testing in situations not under observer's control.
- **ANTS** : An Automatic Navigability-Testing System (for Hypermedia).

Goals of ANTS (1)

- **ANTS** must support Navigability Testing under the Ideal Situation described.
- Observe users while they navigate through hypermedia artefacts.
- Record user navigability behaviour for post-hoc analysis.
- Observation independent of hypermedia device location.

Goals of ANTS (2)

- **Silent Recording:** Users shouldn't notice the way in which they are being observed.
- Moderate communication loads are expected.
- **ANTS** must monitor both large and small Navigation Maps without difficulty.
- Simultaneous Remote Testing of multiple Navigational Maps or prototypes.

Goals of ANTS (3)

- **ANTS** must smoothly run in every kind of hypermedia device available:
 - Stand-alone Multimedia Applications.
 - Multimedia Kiosk
 - Web Sites.
- Easy integration between Automatic Remote Testing and prototypes.

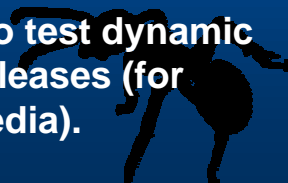
ANTS Development Platform (1)

- Java Technology is one of the best for developing a system such as **ANTS**, due to its many advantages.
- Java's Virtual Machine allows the use of **ANTS** with:
 - Desktop multimedia applications in almost every computing platform (Macintosh, Windows, Solaris,...).
 - Hypermedia Web sites (Netscape Navigator, Microsoft Explorer, ...).



ANTS Development Platform (2)

- Java incorporates many important communication frameworks such as:
 - Sockets.
 - RMI.
 - CORBA.
 - Servlets.
- Servlets might allow **ANTS** to test dynamic Navigation Maps in future releases (for UIMS and Adaptive Hypermedia).



The Anthill Metaphor (1)

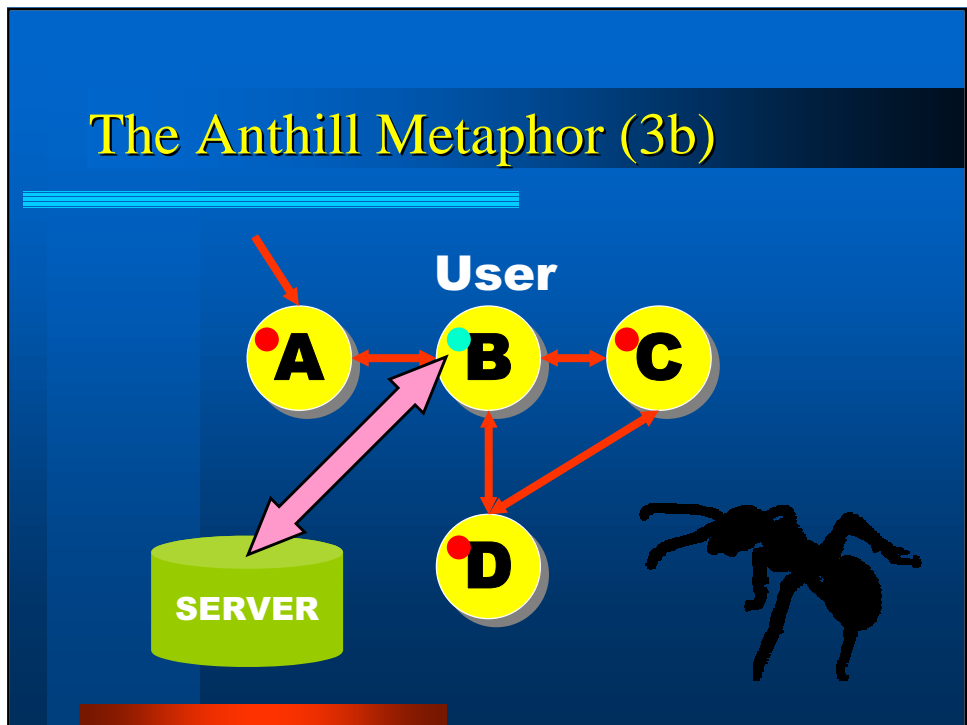
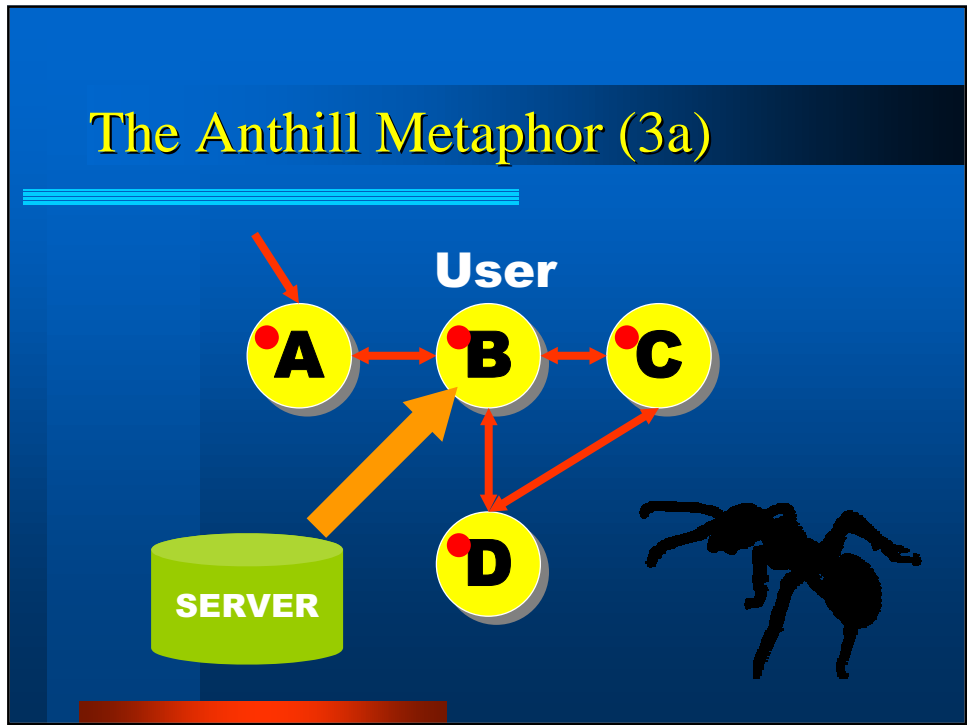
- Based on the client-server paradigm.
- Design metaphor: A community of ants.
 - ants = agents.
 - anthill = server.
 - food = information, user behaviour.
 - picnic = user navigation session.
 - anthill warehouse = databases or files.



The Anthill Metaphor (2)

- For testing, an **ANT** must be included in every node of the Navigation Map to be tested.
- When the user arrives to a single node, the associated ant is downloaded from the anthill.
- Once the **ANT** arrives, it starts to capture data, which is sent to the anthill for post-hoc analysis.





Information Collected (1)

- Complete identification of the node visited by the user.
- Time of arrival.
- Time of departure (time-delay navigation strategies).
- If the SM of the JVM allows it:
 - User ID.
 - Platform used.
 - OS Version...



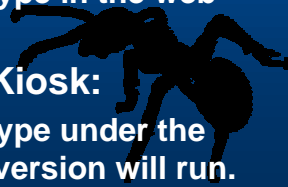
Information Collected (2)

- Depending on the information required by designers, the model can be extended to test HF too:
 - Set of components selected by the user.
 - Window locations clicked on.
 - Visible and hidden components during scroll tasks.
 - User typing skill.
- This information might be automatically processed by UIMS.



Testing using ANTS (1)

- Distribute the prototype among volunteers (as it would be a classic *Beta* version).
- Volunteers will try it at their user-computing environment.
- When testing a Web Site:
 - Testers must install the prototype in the web server.
- When Testing a Multimedia Kiosk:
 - Testers must install the prototype under the same circumstances the final version will run.



Testing using ANTS (2)

- Notice that:
 - Volunteers know they are participating in an Usability Testing.
 - The only fact they ignore is the observation technique used.
 - The technique used will be revealed to users once the experiment will have concluded.



Testing using ANTS (3)

```
DATE: January 16th 1999 (Saturday)
TIME: 17H50' GMT+00:00
```

```
IP ADDRESS: 195.55.11.111 (pinon.uniovi.es)
TIME ZONE: ECT (-2)
```

```
LOGS: (All dates are local to server)
```

```
17:50:45 -> Log in.
17:50:46 -> User arrives to site.
17:51:08 -> User moved to <Computing/InXena>.
17:52:01 -> User abandons site.
17:52:12 -> User moved to <Agenda>.
17:53:23 -> User leaves site definitively.
17:53:23 -> Log out.
```

```
User was here about 00:02:35 seconds
```

Advantages or this Approach (1)

- Navigation behaviour is not altered by external factors.
- Data collected automatically flows from the source to the storage system.
- Testers can focus their efforts into analysing the data obtained.
- There is no need to assign human resources to conduct user observation.



Advantages of this Approach (2)

- Navigation takes place in the user-computing environment.
- There is no need to assign laboratory resources to Navigability Testing.
- The whole process is cheaper.
- More Navigation Sessions can be observed using limited human resources.
- Testing quality improves as it is cheaper to increase the size of the samples.

Advantages of this Approach (3)

- Testing Stage can be extended to the whole product lifetime.
- Support for Adaptive Hypermedia and UIMS.
- ANTS may act as a kind of *GPS* for hypermedia tools.

ANTS may act as a *GPS*



For more information...

- martin@lsi.uniovi.es



Project **TIRSUS**:
www.uniovi.es/~oviedo3/martin



AUTOMATED TESTING OF COM SOFTWARE IN A RAD ENVIRONMENT.

Hugh Newsam

Abstract

This paper describes the results of an investigation into the use of automated software testing methods to validate Component-based software applications. It reviews the unique challenges of such testing and identifies the practical lessons learned concerning the efficient use of these techniques and tools.

1. Summary

Pi Technology is a design-engineering consultancy providing safety-related embedded software applications and supporting tools to the automotive industry. This paper describes the result of the PiVOT project, an ESSI Process Improvement Experiment with the goal of improving Pi Technology's software process through the use of automated testing tools for testing Windows-based software applications (i.e. support tools).

The paper discusses the application of automated software testing to applications developed using an iterative (RAD) methodology and based upon components which are shared amongst several applications, each with their own special requirements and at different stages in the development lifecycle.

The paper considers some of the special difficulties posed by this environment in the application of automated testing tools and how these were overcome. By working with an existing but rapidly developing application, the paper shows how metrics taken at the beginning of the project indicated that the test coverage that could be obtained economically was quite low. During the PIE, the team demonstrated that this can be greatly improved through the use of automated tools but that in order to make their implementation successful, there are many other areas of the process which also need improvement, particularly configuration management.

Finally, the paper reports the quantitative improvement gained and compares the effort required with the testing improvement which would have been gained if that effort had simply been directed at more exhaustive testing of the application using conventional means.

2. Background

Testing software is difficult. Testing software that is being developed under tight commercial pressures and in an environment that is changing rapidly is very difficult. This is the situation faced by many

software engineers developing large applications today in the Microsoft Windows environment and is repeated in many other areas (such as Java-based enterprise-wide implementations).

In order to understand the testing problem, it is necessary to review the trends in software development (for an excellent introductory overview of software development techniques, see for example [1]). Traditional software development techniques have involved a linear process such as the “Waterfall” in which software is developed in distinct stages beginning with requirements, specification, design, code and test. Later variations on this scheme have involved associating testing specification with corresponding stages on the development process; this is normally referred to as the V-Model. For instance, system test is defined at the same time as the functional specification that it is validating.

These methodologies have many drawbacks. They work well for projects in a defined environment, in which all the issues are clearly understood at the outset and where the customer is clear as to what is required. Rarely does modern software conform to this ideal. Instead, the customer has a general idea of what they want to achieve but the best means of implementation cannot be determined without some experiment. This is particularly true when the underlying technology is new or constantly evolving. All of these make complete specification at the outset difficult and attempts to do so often result in the application not meeting the customers (subsequent) needs or projects which miss deadlines due to excessive rework.

Using iterative development techniques such as the “Spiral” model or Rapid Application Development (RAD) can reduce this problem. Applications are developed by defining and creating the framework of a program and then through a process of successive implementation and evaluation, more detailed functionality is added until the desired result is achieved. The attraction of this approach is that changes in direction are highlighted early on in the development process before too much work has been wasted and at each intermediate release there exists a consistent (if restricted) application rather than an unusable portion of the final desired application.

The corollary of this approach is that there will be a larger number of intermediate software releases. Unless changes to the software can be localised, this will result in a massive increase in the testing load, effectively needing a full system test at each stage. In practise, the time for such a system test using manual means may exceed the release interval, undermining the point of using RAD techniques. Instead, software needs to be frequently tested using a combination of regression testing and a much reduced system test or MAT (Minimum Acceptance Test). Nevertheless, for applications which are not rigidly segmented, such testing schemes inevitably mean that intermediate releases have a relative low level of testing leading to problems in the field (sometimes characterised by a feeling that a new release corrected one problem and introduced two more).

More recently, the topic of software reuse has received a good deal of attention and in particular the creation of software as “Components”. These are software objects that have defined interfaces and functionality and can be dynamically linked with applications. Components are only loosely bound to an application and may be shared by many applications simultaneously. In principle, this provides flexibility by which a Component can have its functionality extended without changing the applications that depend on that component. Components are therefore frequently created into libraries that are shared across applications. As long as all existing interfaces and functionality continue to be supported, libraries can be updated without changing the original applications, this is important as co-ordinating updates across different applications is difficult.

Components are created at many levels in the application from very small functions to large blocks of code which are themselves made up of many components. An application such as AutoCal will comprise many hundred distinct components.

Implicit in the component architecture is a much higher testing requirement. When a new component is released, in principle not only does that component need to be tested, but also any application which uses that component. Because the components are bound at runtime, if a computer is running two applications A and B, each of which use version X of a component, simply installing an upgrade to application A which includes a newer release of component X will implicitly cause application B to be upgraded. This is a serious problem. A customer who has validated a mission-critical application B to work in their environment will not want that application changing without their knowledge simply because another application has been installed which uses an updated version of a particular shared component. In an extreme case, for instance if the component is a 3-D graph which is distributed as part of a graphical toolkit, re-testing of all the dependent applications is not feasible and one must rely solely on testing the component itself. This implies that components require rigorous component-level tests in order to ensure backwards compatibility.

One of the principal concerns with automated testing is that as the application changes, so large quantities of test code may also be invalidated. For instance, if an application were being tested by checking the visual appearance on the screen, changing the background colour could make all of the tests fail. If this were to be a regular occurrence, the effort needed to automate software testing would not be worthwhile. Fortunately, the rules of COM come to our aid at this point. They state that all existing interfaces shall be maintained and that a component may only be extended. Therefore, any future version of component X (call it X') must support all the functionality of component X. As a result, creating an automated test for components which act as a validation suite is a very desirable undertaking. These test suites simply need to be extended to include support for the new features of the component as they are developed and no test code should need to be thrown away. It also allows a whole component library to be tested against the validation suite at regular intervals.

This is similar to module testing but takes on a new and increased significance in a component architecture (in this context a module is normally a component which comprises one or more smaller components to implement a logically consistent piece of functionality). In addition, it is necessary to provide a system / integration test of an application to validate its overall operation.

3. Objectives of the PiVOT project

The software to be used as the target for the PiVOT project was AutoCal, a tool for calibrating engines usually in an automotive context. It is a large application built around Microsoft's implementation of component technology and was developed using RAD techniques by which there were incremental releases at short intervals (8-10 weeks). These releases provide incremental functionality and rectify problems found in earlier releases. All the Auto-xxx applications are based heavily on Microsoft component technology using DCOM (Distributed Component Object Model). This is constructed as a library of shared components known as ATOM (Automotive Tools Object Model) developed by Pi Technology, which is used by each application to provide core functionality.

Traditional techniques for testing AutoCal used prior to the experiment, comprised of module level testing followed by system testing using a formal system test plan. A full system test takes several weeks to work through and so, only partial testing was done at each intermediate release. This led to a higher than desired level of bugs finding their way into the intermediate releases and a lengthy period between alpha release and full customer release to allow for the complete system test process. We wanted to determine to what extent we could usefully automate testing for the whole application and the trade-offs between:

- The time taken to create the automated tests;
- The savings in subsequent testing time and improvements in code coverage;
- The code quality which resulted.

We also needed to identify the tools and methodologies for improving and automating testing.

AutoCal is being developed as a core technology from which a number of specific implementations are derived for particular customers. Each of those implementations is likely to require changes to the core components and will be operating on different development lifecycles. This is handled by a configuration management system that allows components to be checked out, modified and returned to the core library. In order to ensure that these components will continue to support all the legacy functionality, we wished to determine whether automated testing could be used to validate components before they were checked into the main tree. In this way, the main tree is “valid” at all times and at any point a full customer release based on the current iteration of components can be made.

Finally, we wanted to develop an extension to our present software development process, which would ensure that components and applications would be adequately tested. We wanted to provide guidelines to developers that would help them write applications in a way that made them amenable to subsequent automated testing.

4. Tools to achieve objectives

To achieve the objectives tools are required. An automated testing tool is needed so there is some platform on which to develop automated test scenarios. Secondly, a coverage tool is required so that the effectiveness of the baseline and then the automated testing can be quantified.

4.1. Selection of the automated test tools

Automated test tools work by allowing the user to emulate operations such as keystrokes and mouse-clicks using a scripting language. Scripts can then, once produced be used over and over again. The testing tool will generally offer the service of producing these scripts based on the input to some sort of macro recorder. Testing tools also incorporate the ability to retrieve and validate information to determine whether the application has reached a particular state. There are several techniques offered by the testing tool to retrieve information from the application under test. All of these techniques require that the information of interest is identified and then retrieved, that information can then be compared with the expected result to generate a pass/fail condition. Gathering information such as text

from edit boxes and list controls is not too difficult, although there are some issues involved in finding the appropriate control in the first place. Other information such as bitmap displays is not so easy to validate and the tools provided by the testing tool for doing so are generally less than satisfactory.

The requirements of a tool to test components (which may not have a user interface) and an overall system are somewhat at variance. However, for consistency it was desired to have a tool that would support both.

A key requirement is that the tool can manipulate the application under test. In the case of the system test, recording simple user interface commands in the form of a script and validating application output can do this. With component technology it is possible to interrogate the properties of the user interface components themselves. This provides a much more robust means of validating the state of the application as these tests are now insensitive to system-wide changes such as screen resolution. Before this could be done in the AutoCal system test, it was necessary to extend the “container” application that holds the AutoCal components so that it would provide an automation interface and permit inspection and manipulation of the properties and methods of the components. This was a significant and unexpected undertaking and took several weeks.

Testing the components themselves is slightly different since many do not have a user interface. As a result, it is necessary to write a simple “wrapper” application (for instance using Visual Basic) to provide access to the component itself. This wrapper becomes part of the test suite and is maintained with the code.

As the company has already had experience with the Rational range of automated testing tools it was decided that we would evaluate their Visual Test package and their SQA Suite. Taking into account that the latter is approximately five times more expensive.

The environment in which the testing needs to be done required that there be a strong degree of programmability behind the testing tool. For example, if the test involved manipulating variables represented in a tree control, there may be one thousand leaves in the tree control. Each of these leaves must be dragged and dropped onto another control somewhere else in turn. It is of course possible to do this using a ‘macro recorder’ style method but would become very tedious when a ‘for...next’ loop would have done the job. There are many situations in testing this particular type of application where this type of recursive or repetitive action is required.

Decision-making is also important, if we take the example of the tree control once again then it may be required that leaves of certain types be dragged to one place and other types to another place. Had these actions been carried out using the macro recorder style method of generating test scripts it would be very difficult to go back and change the script if something had changed, e.g. maybe the way the types in the tree are described has changed. This point is made because the two tools really take one or the other approach. Both tools will let the user enter code and both will serve as an advanced macro recorder / validation tool, but neither are good at both.

It was decided that we should use Visual Test because it offered more flexibility in allowing the user to write scripts rather than record them. For the types of reasons outlined in the previous paragraph writing scripts rather than recording them was very important. SQA Suite was far superior when it came to recording scripts using the Robot application supplied, but as we were not going to use this tool extensively, the extra cost of SQA could not be justified when the programmability was no better if not worse than that of Visual Test.

4.2. Selection of the code coverage tool

There are three main types of coverage metrics considered during this experiment. Firstly, line coverage, this simply tells the user what proportion of the total lines of code have been executed. Secondly, function coverage, this is very similar to line coverage and shows the user the proportion of the total functions in the application that have been executed. Once again the user can usually view the specific functions covered or not covered during execution. The third type of coverage differs from the other two. Branch coverage, this requires the analyses the structure of the application or module and the total number of paths through the code. The tool then measures how many of those paths have been taken during execution.

The actual decision as to which tool to use was limited by both cost and convenience. Both coverage tools we looked at offered only line and function coverage. No tools offering branch coverage were investigated as such tools were beyond the scope of the project's budget. As both tools offer the same in coverage information the only real issue in choosing one boiled down to convenience. We wanted a tool that we could use on a debug build of our application with a minimum of effort required in getting the tool to function correctly. Also we needed to be able to integrate this tool with the automated testing tool we had already chosen. It was for these reasons that we chose to use Rational's Visual Pure Coverage. The tool requires that no 'special' build of the application is made and so meant no changes to our extensive, current build procedure. Secondly the ability to turn the coverage tool on and off from the testing environment made the tool more attractive.

5. Baseline

5.1. Coverage

In the starting scenario, components were not tested individually but only by implication as part of a MAT. This comprised two elements: a minimum acceptance test that is undertaken frequently and a full system test that is only undertaken at major releases. These tests are fully manual (though defined in a test plan and therefore repeatable) and the code coverage obtained when running through these tests is shown in Figure 1.

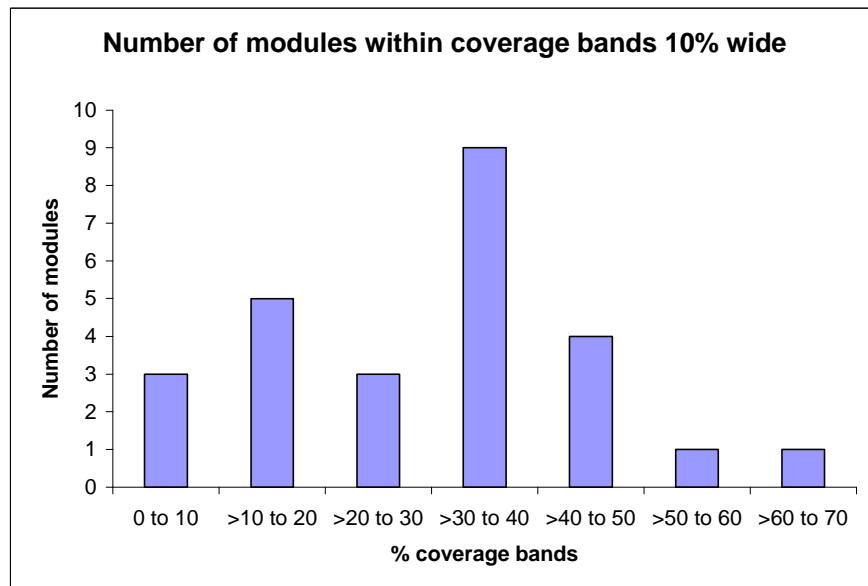


Figure 1

Overall the average function coverage was 25.4%. Figure 1 shows the number of modules within coverage bands 10% wide. We obviously need to look at the parts of the system with the lowest coverage and for those areas identify the modules in greatest need of attention.

As can be seen from the data, the overall code coverage is low. This is because most test paths execute largely the same overall code and it is very difficult at the system level to obtain significant coverage of the individual components.

There are eight modules in the application with coverage of 20% or less. It is clear that these are more likely to contain undetected bugs. One of these modules in fact is also one of the largest in the application, containing almost 1/3 of all the application code. It is clear that our attention should be turned towards this module.

For the purpose of this experiment we concentrated on the one large module. As this module was at the heart of the application the quality of the product would gain from the improved bug detection. The module is large enough to provide us with enough data to go away and make some predictions based on the results.

5.2. Analysis of code quality

By analysing the change requests, it should be possible to create a metric for overall code quality. This is a qualitative measure only and its relevance is restricted to the application in question i.e. it is possible to use the metric to indicate whether the code quality is improving but it would not be possible to compare directly the metrics between two dissimilar applications.

The change request tool used on this project does not make the collection of CR data very easy. Data about which particular module the bug appeared in is not stored in the database. Because we have concentrated on a single module during the experiment this is a major drawback, although rate of CR generation is available for the application overall this is influenced by so many factors outside the experiment the data becomes irrelevant.

6. Implementation

6.1. Component-level testing

Initial testing at the component level is undertaken as follows: A representative selection of components was selected, ranging from those with a high graphical content to those that simply manipulate data. For each component, a suitable test harness was written and some automated scripting developed. Such testing was successful and the code coverage of the testing rose dramatically as a result of these individual tests. However, it did highlight some points:

Generally the automated testing tools do not provide very clean interfaces to the COM objects within the AutoCal package, especially the non-visual components. Visual Basic provides a very clean interface to COM objects from the developers' point of view. Although idea of abstraction at the COM interface level may not add to the effectiveness of the testing it is useful in reducing the time spent on changing the test should the interface change.

In order to test components in this way, a complete and relatively formal specification of the interfaces and functionality of the component is needed as well as an overall design documentation which describes in detail the relationship between these components in the end application. For a software application that is made up purely of an interlocking mesh of software components, particularly if it is developed under time pressure, this degree of definition is not always provided. Therefore, a much higher level of design documentation is necessary in order that those creating or subsequently maintaining the tests are able to create representative scripts.

The result of this phase of the testing was to re-define the software process such that all components are treated as applications in their own right with a formalised specification and design documentation. The development process has been extended so that the original developer is now responsible for delivering not just the component software but also a suitable wrapper application and the automated test code which both tests the functionality and meets the desired code coverage metrics. This represents a major shift in attitude in the company by which the developer is responsible for the entire lifecycle of a component and not just the design and implementation phase.

6.2. System level testing

At a simplistic level, system testing is a straightforward process. A test plan is written and then executed manually while the test tool records the actions. These actions can then be played back in the future to re-run the test. However, AutoCal is a core technology that is provided in customised form to several customers. Each customer has different requirements and a different release schedule and will

need extensions to the core AutoCal library. Therefore, the base AutoCal is a constantly moving target and this makes it difficult to create meaningful experiments in this environment.

AutoCal is kept in a configuration management system. Normally a build of AutoCal is made from the “tip” of the build tree, i.e. the most recent version of each of the components. For the PIE, a line was drawn through the AutoCal configuration system which defined the “master build” of the software and so fixed the state of the software at one particular time. The AutoCal application would continue to be developed but it would not affect the master build. Once a baseline system test has been created, the line representing the master build can be moved progressively, taking in more and more recent versions of components. In this way, the testing is grown incrementally and in a controlled way, which was the intention of the project.

The component-level procedure described above was used to create criteria for when a component could be merged back into the master build. In this way, the probability of the introduction of a modified component “breaking” the master build was minimised.

Rather than creating the system level test as a single monolithic entity, it is attractive to make it follow more closely the structure of the application itself with a series of quasi-independent scriptlets. These are then built into a library of tests that can be mixed and matched depending on the composition of the final application.

7. Results

7.1. Coverage data

The results of the testing are shown below. The individual tests on the components improved the code coverage dramatically, providing a basis for future validation of enhanced versions of the components. The overall coverage at the system level also improved. The change request data is less clear cut.

During the experiment we concentrated our efforts on a single module at the heart of the application. This module constitutes approximately 1/3 of the total application code and so is large enough to produce some meaningful results.

The histogram in Figure 2 shows line coverage data for the individual source files constituting the single module used in this experiment. The histogram shows the number of source files in the module falling within evenly spaced coverage bands. The before values represent the coverage this module received during manual testing before the implementation of the automated testing. The after values represent the coverage received using the automated test scripts developed during this experiment.

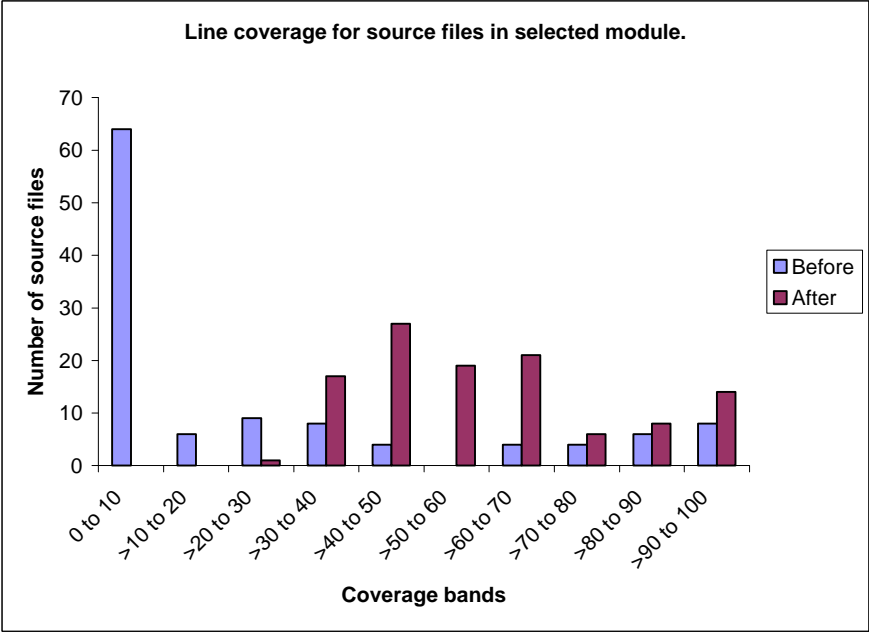


Figure 2

Figure 3 shows a plot of baseline coverage against the proportional improvement after automated testing for the source files in the module used. The baseline coverage value is the line coverage achieved during manual testing before the automated testing was introduced. The proportional improvement is the factor by which the coverage has increased after implementing the automated testing.

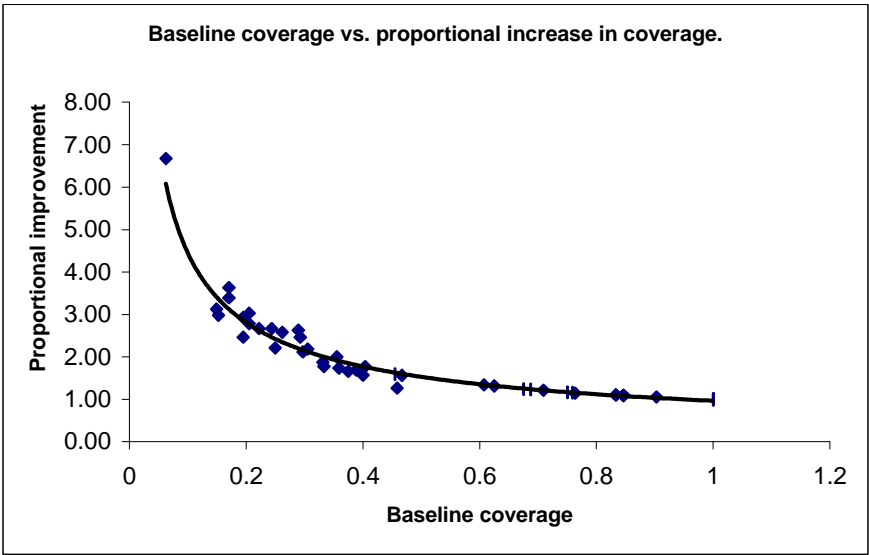


Figure 3

This plot shows that if the initial coverage is low then applying some amount of effort is going to have a large impact on the coverage of the code. As the baseline coverage gets better it becomes increasingly difficult to improve the coverage any further.

Given this information it would be beneficial if while trying to improve coverage of an application the coverage were increased in stages. Aim to take one module and increase its coverage level to some value still below the final desired level and then move on to the next module. Once all modules with coverage below the first stage have been addressed increase the target level and go around the loop again. This way the effort in the beginning is going into dramatically improving the coverage of the application in all areas of need quickly. If each module were to be brought up to 80% coverage before moving onto the next then much time would be spent on the later development of the testing of the first module and in this time the coverage of the rest of the application would still be very low. This could be thought of as a kind of RAD approach to testing.

Figure 4 shows a plot of the baseline coverage against the coverage after the implementation of the automated tests. The many data points at the origin of the y-axis represent the source files in the chosen module, which received no coverage before the implementation of the automated testing.

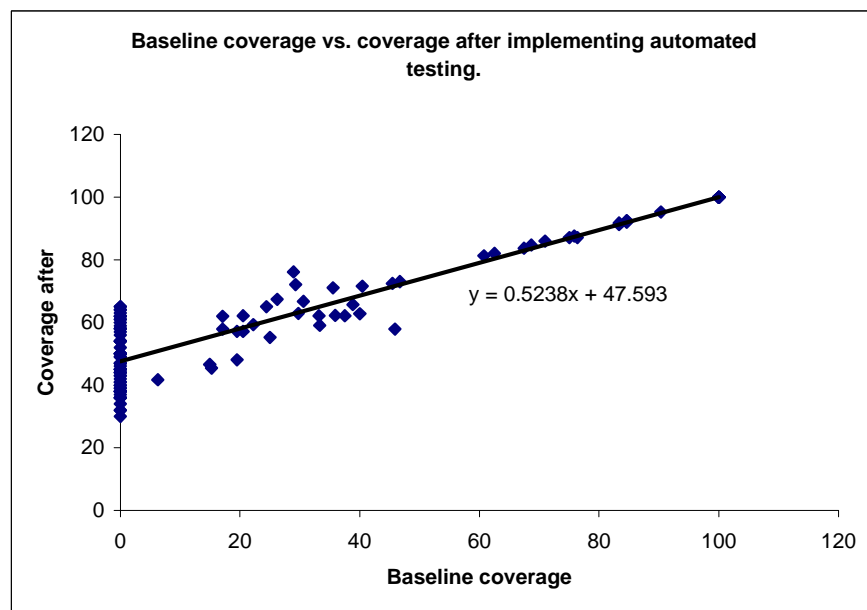


Figure 4

Given the data in Figure 4 we can use the trend line to predict the level of coverage that could be achieved if a similar process is applied to all other modules in the application. Because the module used in the experiment is so large it gives us a reasonable number of source files with a ranging level of coverage from which to perform our extrapolation.

Figure 5 is a histogram showing the results of the extrapolation performed using the data presented in Figure 4. The before values in Figure 5 represent the number of modules in evenly spaced bands of coverage at the baseline. This data is the same as that shown in Figure 1. The after values shown in Figure 5 are the predicted values for all the modules in the application if a similar process is applied to those modules as was applied to the module used in the experiment.

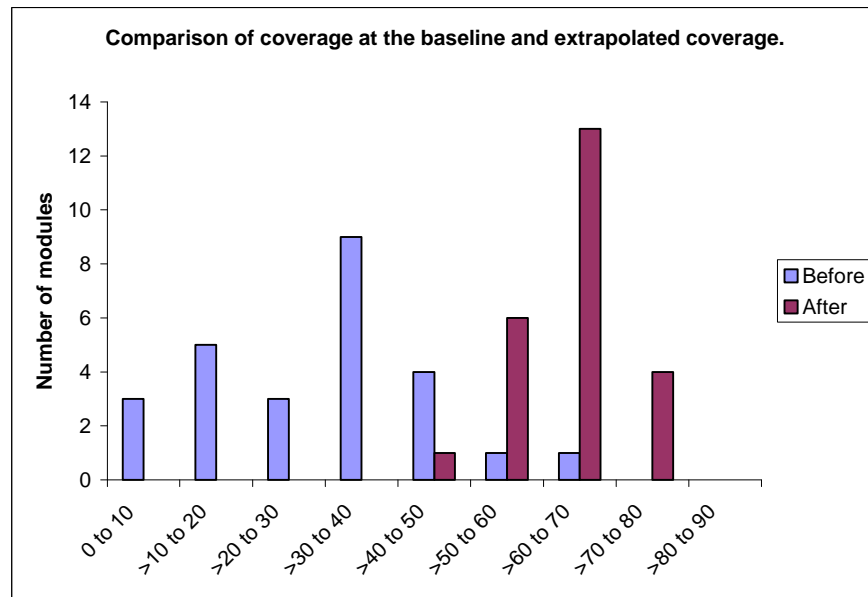


Figure 5

7.2. Comparison with conventional techniques

If the resource used to develop the automated tests were put into developing manual tests and performing these tests at each release then initially the testing would be improved. However automating these tests has provided a means of continuing these levels of coverage after the effort has been put into their development. Certainly the initial improvements in the testing would also be greater than these final results but again this improvement would be temporary. Once the effort is removed the testing falls back to where it was (as there is no time to perform the testing manually with no additional resource – the original problem). A lot less effort will be required once the tests are developed maintaining the tests than would be required continually performing the full testing manually.

8. Conclusions & Recommendations

8.1. Technical impact

The impact of the project has been greater and more wide-ranging than had initially been anticipated. The need to be able to test in an automated manner has caused a re-evaluation of the whole software development process. In particular, it had highlighted some of the problems with component based software development (at least as defined by Microsoft) and caused us to take very seriously the issues it raises.

The immediate result has been:

- A much clearer definition of the structure of the test application. Object models are now defined more rigorously with object oriented modelling tools such as Rational Rose from Rational Software.
- Much more concentration on the importance of good configuration management
- Much more attention to abiding to the rules of COM, never destroying existing interface but only extending them in order not only to preserve legacy functionality but also to continue to support older automated test cases.
- Developers of new components are more considerate of the testability of the objects they produce since they may now have also to deliver wrapper applications and test scripts with the software. An example of this is the need to make the internal data available through an interface to enable testability.
- If automated tests are developed alongside components this may lead to more testable components being produced and provide a mechanism for integrating components into the master build.
- Structuring the tests into a library of “scriptlets” which are aligned to the structure of the application being tested helps to minimise the rework needed to the test scripts when applications change.
- More rigorous regression testing forces developers to take more care to document as they design (and therefore potentially catch problems earlier) rather than fire-fighting at a later point.

8.2. Key Lessons

The main lessons have been:

- You cannot just system test and expect to get reasonable coverage. Module tests must also be done at the component level. This puts a greater onus on the system architect to define the project in detail at this level.
- Configuration management is essential; otherwise you are always building on sand. Similarly, the way in which components are extended needs to follow strict rules if previously developed tests are not to be invalidated.
- Testing metrics are very difficult to gather. A different testing process changes many variables at one go – time, coverage, location in the cycle etc. It is therefore difficult to perform a controlled experiment to show quantitatively the benefit. However, subjectively, such testing techniques are regarded within the company as essential for component-based software development.
- To test software using automated testing tools the software has to be designed in a testable fashion. Much of the work of developing automated tests would have been avoided if the software had been written with more consideration for automated testing.
- Automated testing is still difficult when it comes to validating visual aspects of an application. It is possible to test that the internal values have changed in a visual component but it is much more difficult to check the image has actually been updated on the screen. This is left to Bitmap comparisons with current testing software but the results from these comparisons are usually less than satisfactory.

- Component technology is a powerful tool but not a panacea. In particular, the cost of managing a project based on components is significant and has to be weighed against the savings that arise from reuse of development effort.
- The automated testing forces a greater degree of documentation of the software. This documentation can then be used effectively in the marketing process to develop new business.
- The change management tool used on this project is limited in the way bugs are reported. Not enough information is entered into the database to describe where in the development process the bug was introduced and where it was detected. The tool also provides no information as to which module the bug appeared in, this information needs to be available in a field in the database so it can be filtered on. The quality of the information the change request system was very poor and an ongoing effort is being made to improve this so better analysis can be carried out in the future.

8.3. Conclusions

The experiment has had a more fundamental impact than had initially been expected. Not only did it highlight the issues of testing but it also demonstrated a very high dependency on detailed specification and configuration management. The automated testing procedures have been instrumental in the creation of a process to manage a core library of components, which is being developed simultaneously with the dependent applications. In addition, it has resulted in a significant improvement in the overall development process.

The use of such techniques places challenges on the overall software development process and is likely to be successful only in an environment in which the overall software process is well defined and controlled.

9. References

[1] McConnell, Steve; Rapid Application Development, Microsoft Press 1996

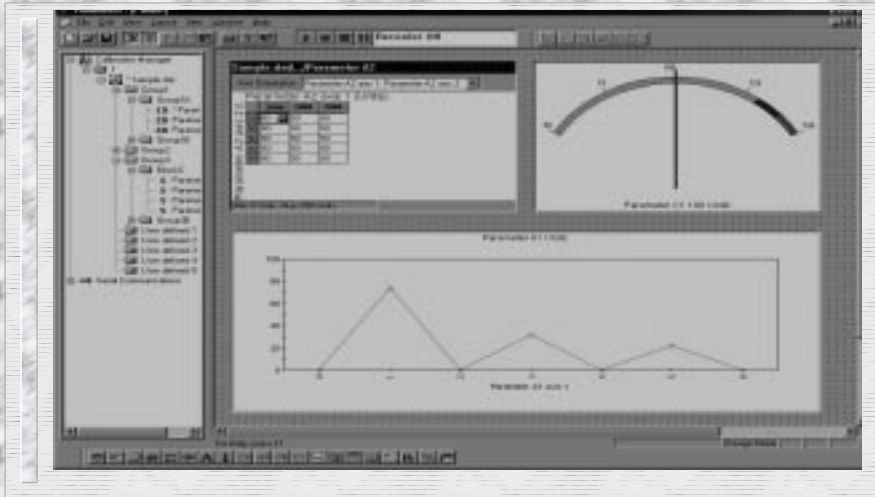
PiVOT

Automated Testing of COM Software in a RAD Environment

Background

- Pi Technology.
- Automotive tools and controllers.
- Embedded programming environment.
 - Customer understands requirements;
 - Safety critical so 100% test coverage.
- Windows programming environment.
 - Customer is unsure of requirements;
 - RAD development environment.

AutoCal



Project Inspiration

- Customer unsure of requirement so product requirements change frequently.
- RAD development so releases were happening regularly (every 8-10 weeks).
- Developer resources are limited.
- System lacked in depth testing at each release.
- ESSI funding was available.

Project Goals

- Improve the level of testing that can be obtained at incremental releases.
- Reduce the requirement for developer resources in the long term.
- Try to this by automating testing.
- Gather information useful to other projects.
- Disseminate any information gathered by the project.

Tools Needed — Automated Testing Tools 1

- What do they offer?
 - Manipulation of Windows based applications;
 - A means of logging results;
 - A script language for generating test scenarios;
 - Script generation tools.

Tools Needed - Automated Testing Tools 2

- What Choice was there for us ?
 - Rational Visual Test
 - Rational SQA Suite
- Factors influencing decision.
 - Cost, effectiveness, programmability.

Rational Visual Test

The screenshot displays the Rational Visual Test (RVT) application window. The title bar reads "ToolBench1 - Rational Visual Test". The interface is divided into several panes:

- Left Pane (Project Explorer):** Shows a tree view of the test suite "ToolBench1". The tree includes a "TestBench1" folder containing sub-items like "TestBench1.tst", "LoadD.tst", "OpenApplication.tst", "OpenDatabase.tst", "OpenDatabase.tst", "DisplayOf.tst", "ToolBench1.tst", and "ToolBench1.tst". Below the tree are buttons for "Include" and "Exclude".
- Right Pane (Code Editor):** Displays the source code for a test case named "ToolBench1.tst". The code is written in a language similar to Visual Basic or VBA, featuring declarations for variables like "strWindow" as string, "lcnm" as long, "lcnm2" as long, "lcnm3" as long, "lcnm4" as long, "lcnm5" as long, "lcnm6" as long, "lcnm7" as long, "lcnm8" as long, and "lcnm9" as long. It also includes comments: "DIE statements, TYPE declarations, etc constants, and so on for test case".
- Bottom Pane (Output Console):** Shows the execution results of the test. The output text reads: "ToolBench1 opened", "3000", "The number of items in this item = 2", "Item No 1 is called Calculator Manager", "This is Item Number 1", "Cal Manager item is selected", and "Operation Successful".

The status bar at the bottom indicates "Ready" and provides keyboard shortcuts for "View", "Exit", "Print", "F1", "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9", "F10", "F11", "F12", "Alt", "Ctrl", "Shift", "Enter", "Esc", "Tab", "Space", "Back", "Forward", "Home", "End", "Up", "Down", "Left", "Right", "Print", "Print Screen", "F1", "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9", "F10", "F11", "F12", "Alt", "Ctrl", "Shift", "Enter", "Esc", "Tab", "Space", "Back", "Forward", "Home", "End", "Up", "Down", "Left", "Right", "Print", "Print Screen".

Tools Needed — Coverage Tools

- What do they offer?
 - Branch coverage;
 - Function coverage;
 - Line coverage.
- What choice was there?
 - Rational Visual Pure Coverage;
 - NuMega True Coverage.
- Factors influencing decision.

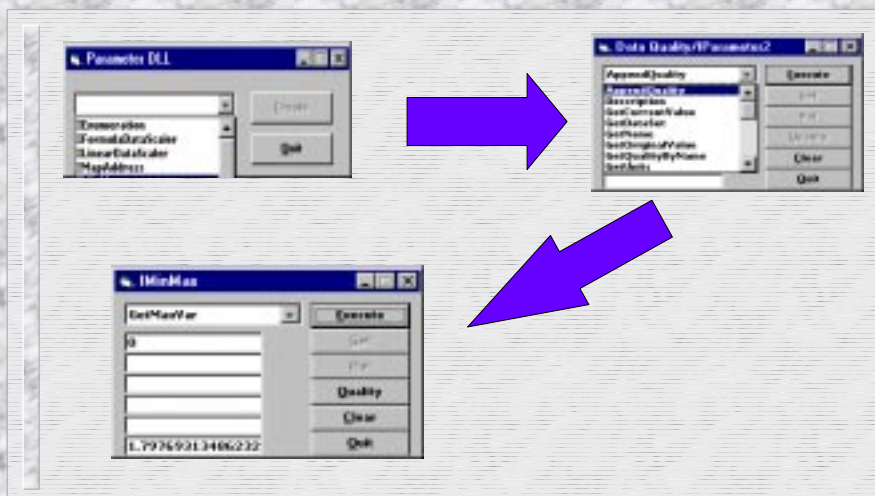
Problems Faced

- Limitations imposed by testing tools.
- Effect of these limitations on the software under test.
- Problems with coverage tools.
 - Primarily integration into the build process.
- Invalidation of test scripts.

Issues Arising from Automation of Testing

- COM interface publishing rules allow for effective regression testing.
- Test environment COM support.
- Wrapper applications required.
- Identification of windows and controls.
- Support for automated testing from the application.

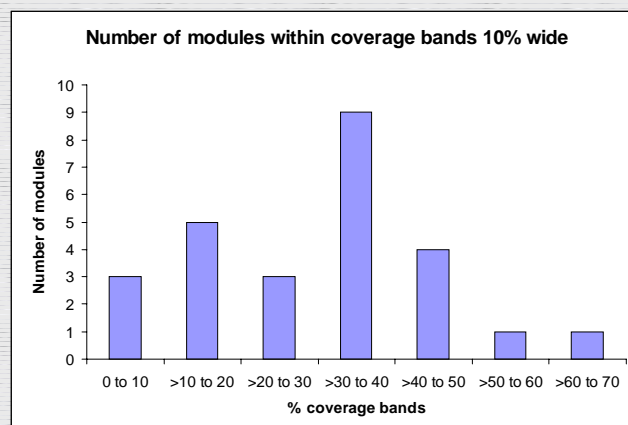
Test Harnesses



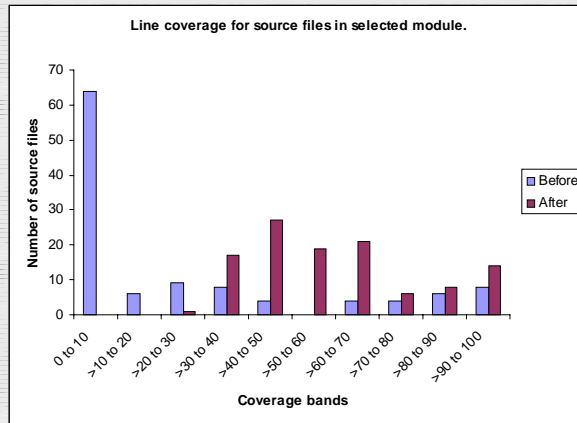
Results of the Testing

- The following are the results of the improvement of testing.
- Baseline coverage.
- Improvements for parameter.dll
- Extrapolation of results over the whole project.

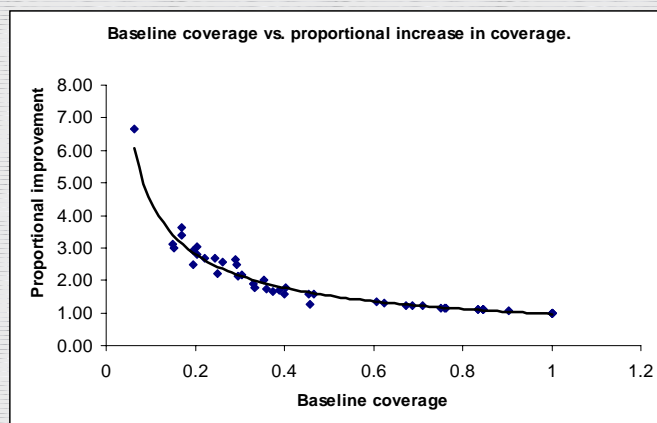
Results of the Testing — Baseline Coverage



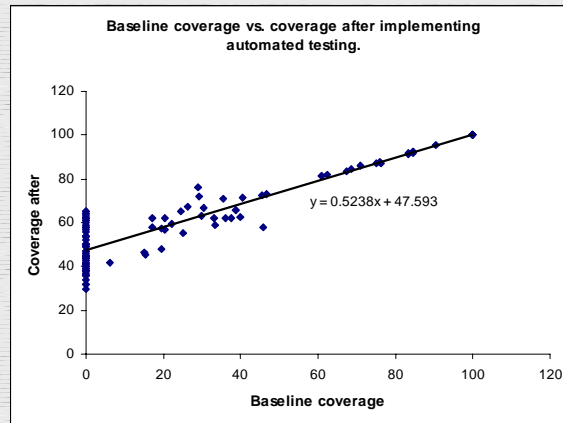
Results of Testing — Coverage for parameter.dll



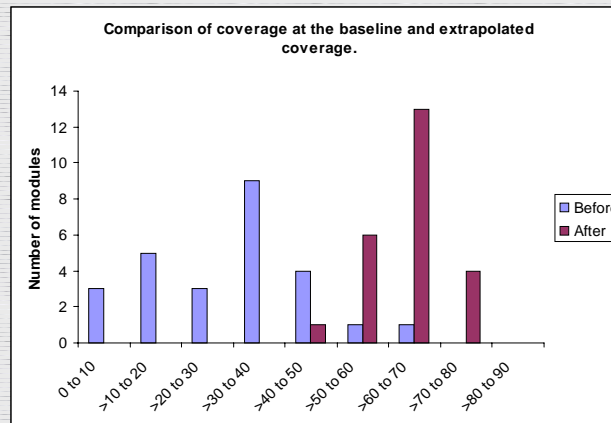
Results of Testing — Proportional Increases



Results of Testing — Extrapolation Data



Results of Testing — Projected Results



Technical Impact

- More effective regression testing forces more care on the part of the developer.
- The importance of configuration management.
- Rules of COM interface publishing are now even more important.

Technical Impact

- Developers concern with testability.
- Parallel development of modules and test scripts is now desired.
- Reduction in size of individual test scripts.
- Just as Microsoft have been doing now for years, we now build and test almost every night.

Lessons

- Rules of COM interface publication impact more widely than originally thought.
- To perform good automated testing software needs to be developed with this in mind.
- By applying the above good code coverage can be achieved.

Lessons

- Validating visual aspects of an application still remains a problem - resource intensive.
- The change management tool used on this project proved to be inadequate.
- Automated regression testing is valuable.
- This is the way forward

Actions

- Project Quality Plans all updated to promote use of test harnesses / testability.
- Test Bed consisting of 5 PC's has been set up. Engineers can access these directly or remotely by PCAnyWhere
- Vacation Student hired to Automate ALL AutoCal MAT and System Tests. Looking for Full Time testers.

Example New Test Harness

The screenshot displays the AUTUM Harness software interface. The window title is "AUTUM Harness". The main area shows the following configuration:

- Default:** C:\PROG1\2002\DOWN\AUTUM\TEST\... (with a "Load Default..." button)
- Parameter file:** Name: Parameter A1, Tag/Name: A1, Description: Parameter A1
- Options:** LHM, Fixed step, LHM (with "Previous" and "Next" buttons)
- Buttons:** Done, Copy to..., Compare with..., Automatic, Goto...
- Quality:** Data Quality, Shape Quality, Map Address Quality, MinMax Quality, Security
- Units:** Min: 0, Max: 100, Counts, Value: 0, Depth: 0, Rate: 0 (each with a "Set" button)
- Micro map:** 01000000 (with a "Clear" button)
- Major step:** 1
- Units:** Units: Links
- Position:** None given
- Buttons:** Display Test..., View Data..., Load, Goto, Bulk, Auto for

Any Questions ?



Testing C++

Why It's Hard and How To Do It Well

Misha Dorman

IPL Information Processing Limited
Eveleigh House
Grove Street
BATH BA1 5LR

mishad@iplbath.com <http://www.iplbath.com>

Abstract

Object-oriented technologies, and the C++ language in particular, have enjoyed great popularity in recent years. Proponents have promised many benefits, including reduced lifecycle costs, increased reliability and improved maintainability. Unfortunately, not all of these benefits have been fully realised. One area where OO methods are relatively immature is testability and testing.

The naïve application of traditional testing techniques to object-oriented C++ software has not been completely successful. Isolation testing – a technique successfully used to “divide and conquer” traditional testing problems – becomes unmanageable when applied to OO software systems.

This paper explores the problems from a practical point of view, and discusses an improved integration testing approach based on an extension to the widely used source code instrumentation technique. This approach provides many of the advantages of isolation testing without forcing the tester to laboriously write simulated code (“stubs”) for the rest of the system.

Measuring test effectiveness of OO software using traditional coverage and complexity metrics can be unreliable and misleading. This paper discusses some metrics which provide more insight into the static and dynamic complexity of the software under test. In particular, they attempt to measure the complexity of the interactions between classes and the extent to which the polymorphic features of the software have been exercised. In addition, modifications are proposed to the traditional coverage metrics, to make them more appropriate to C++ software.

By combining the enhanced integration testing techniques and new metrics described in this paper, together with appropriate tool support, C++ testing will be more efficient and more effective.

1. Testing As We Know It

A vast amount of experience has been gained in the testing of software systems. Most of this experience applies to software written using procedural languages (C, Ada83), using top-down design methods.

Typically, the focus of testing is dynamic testing – the software under test is executed, and the actual behaviour compared with the expected behaviour. A dynamic test harness is used to drive and control the test: it sets up the input data required for the test, invokes the software under test, and compares the resulting output data with the expected values. Dynamic testing is performed at all levels from unit, through integration, to system-level testing.

During unit testing, isolation techniques are often used to “divide and conquer” large testing problems. Through the use of stubs, individual units (typically functions) are tested in isolation from the rest of the system. Isolation testing allows units to be tested before they are integrated with the rest of the system¹. When units are subsequently integration tested, test effort can be concentrated on the correct operation of the interfaces between units. The use of stubs (which form part of the test harness) also has benefits during maintenance: re-testing at the unit level is limited to those units which have actually changed. It is not necessary to re-test all the code which depends on the changed units.

At all levels of testing, but particularly at the unit and integration levels, structural coverage analysis methods are used to measure the quality of the test. During unit testing, statement, decision and condition coverage metrics are used to ensure that all parts of the code have been exercised. During integration testing, entry-point and call-pair coverage metrics are used to ensure that the interactions between units have been fully exercised.

It is an unfortunate fact of life that there is never enough time to do everything. This applies to testing – we often simply cannot afford to test every single component. Static structural complexity metrics are calculated and used to estimate defect density as well as coding, testing and maintenance effort. These estimates are then used to focus the available testing effort where it will produce the most benefit. This process can be formalised, based on an ongoing metrics collection and calibration process, or informal, using the complexity data to identify “outliers” which are worthy of further investigation.

2. Why Is Testing C++ Harder?

The C++ language is considered by many to be “merely” an extension to the C language. Some practitioners have claimed that testing C++ is no different from testing C - simply redefine the unit from function to class and carry on as normal. It is true that the traditional approaches to test case design (error guessing, domain coverage, state-based testing and others) are still useful and valid. However, attempts to apply traditional testing approaches to C++ software have encountered significant difficulties. Some of these difficulties stem from the use of object-oriented design methods; others from intrinsic features of the C++ language.

2.1 Too Many Dependencies

Typical object-oriented designs consist of a large number of interacting components, each relatively small by itself. This is an important strength of OOD – it provides the power to break down large, complex problems into smaller ones.

¹ even before the rest of the system is *written*. This allows much greater flexibility in the allocation of work; it is not necessary to complete coding before unit testing begins.

An unavoidable consequence of such designs is a massive increase in the number of dependencies between units. These dependencies include containment, inheritance, reference, parameters, and return values.

Attempting to use isolation techniques during unit testing of most OO systems is bound to fail. The overhead involved in the creation of test stubs for every dependency is simply too great.

As if these problems were not enough, features of the C++ language create even more dependencies. Class implementation details are exposed in header files. Although clients are prevented from taking advantage of this, it causes significant problems during isolation testing.

In order to stub a constructor for a class, it is necessary to provide initialisation values for all data members of the class² - including the private ones³. However, the stub is not part of the class (it is part of the test for some other class), and as such should have no knowledge of, nor access to, the implementation details.

Remember, one of the reasons for isolation testing was to reduce the re-testing effort during maintenance. When constructors are stubbed, this benefit is lost: changes to thousands of stubs, in the unit tests for hundreds of different classes, may be required whenever the implementation of a class is changed. Even if the class didn't have any problematic data members when first written, they could be added at any time during maintenance.

Clearly, this situation is unacceptable – we have been forced to forego data hiding and abstraction. With them goes one of the main benefits of OO, the ability to change the implementation of a unit without requiring changes to its clients.

2.2 Integration Testing

We have seen above that isolation testing is inappropriate for most OOD/C++ systems. The alternative to isolation testing is bottom-up testing. First the lowest level units are tested (those that have no dependencies on other units, and therefore do not require isolation). Then, the units which directly depend on the lowest level units are tested in integration with the already tested units. In this manner, testing proceeds up the dependency hierarchy, until the complete system has been integrated and tested.

At each level of bottom-up testing, the test harness must target both the low-level structural faults normally associated with a unit-level isolation test, and the interaction faults normally targeted by an integration test. This combination makes bottom-up testing more complex than either unit or integration testing alone.

The increased size of the “lump” which is the software under test makes it difficult to exercise all parts of the unit's code. The influence that the test harness has over the software under test is limited to defining an execution environment and passing in parameters. As testing proceeds to higher levels, the “already tested” units start to get in the way of the unit testing of the highest level units – sanitising the input environment and making it impossible to test obscure cases or error handling code.

In typical C++ systems, these problems are exacerbated by the use of data hiding and the presence of circular dependencies between units. Data hiding makes it even more difficult for

² More accurately, all the data members which are of class type, and which do not have default constructors. There is often at least one such member - and as we shall see, even one is too many.

³ Of course, usually all the data members are private.

the test harness to force the execution of “difficult” cases: it is prevented from setting the private data, or calling the private member functions, of the class under test. Circular dependencies force units to be integrated in groups instead of individually, making effective unit testing even harder to achieve.

2.3 Re-use

Much of the hype surrounding object-oriented development has focused on “re-use”. This overloaded term can apply to the re-use of code, designs, or architectures; here we are concerned primarily with the re-use of code.

Code re-use predates OOD, of course. Every time a programmer uses a library function, they are “re-using” code. Such re-use is limited, however, to that which is available in libraries. What is different about re-use in OO systems is that code is re-used in a way that provides different or additional functionality, without duplicating the common functionality. This provides the flexibility which enables large-scale re-use of components. In C++, re-use is provided through inheritance and template instantiation.

When a class inherits a member function from its base class, it is re-used in a new context. The behaviour of the inherited member can change, for example if it calls a virtual member function which has been overridden in the derived class.

When a template is instantiated on a new type, it is also being re-used in a new context. The behaviour of the template can change because its use of the operation “+”, for example, means different things when applied to different types.

Re-use through inheritance is a dynamic, run-time feature. The behaviour of the inherited member function depends on the actual (dynamic) type of the object to which it is applied. Re-use through template instantiation, on the other hand, is a compile-time feature⁴. The behaviour of the template depends on the static (compile-time) type on which it is instantiated.

In both cases, the behaviour of the re-used code is (potentially) different in each new context. In [Harrold], criteria were defined for the re-testing of re-used software. In essence, re-testing is necessary if the behaviour of the re-used component is changed⁵ in the new context.

Every time code is re-used instead of written from scratch, effort is saved. However, testing may account for 30-50% of development effort. If a new test script is required whenever the unit is re-used in a new context, then the savings obtained through re-use of code will be significantly reduced.

2.4 Coverage

Typical OO systems consist of a large number of relatively small member functions. Applying traditional coverage metrics to these systems is certainly possible, and necessary. However, achieving 100% decision and condition coverage during a class test is usually easier than achieving the same level of coverage on the equivalent system written in a procedural language,

In an object-oriented system, an important part of the control flow arises from polymorphism (virtual functions in C++). In an equivalent procedural system, this control flow would probably have been written as explicit `if` or `switch` decisions - and would be targeted by

⁴ In typical C++ implementations, re-use via templates results in duplication of object code. However, this is purely a compiler issue - the important thing is that the source code is re-used, unchanged, and automatically.

⁵ Or potentially changed.

traditional structural coverage techniques. In the OO system, this control flow is hidden from the coverage analysis – because the definitions of the coverage metrics are too restrictive.

As well as “missing” polymorphic control flow, traditional coverage metrics fail to take advantage of design information made explicit in OOD which can enable more advanced coverage analysis. In particular, many classes can be modelled as state machines. The potential states in which an object can be, are as important a structural characteristic as the potential branches that can be taken in a member function, and as such should be measured as part of coverage analysis.

Clearly, new coverage metrics are required if we are to benefit from structural coverage analysis techniques.

2.5 Complexity

Just as traditional structural coverage metrics are insufficient to measure the dynamic complexity of C++ software, traditional structural complexity metrics are insufficient to measure its static complexity. The complexity of the interactions between classes, including the inheritance and polymorphic relationships, is clearly an important factor in the readability and comprehensibility of C++ software – and can therefore be expected to have an impact on defect density as well as coding, testing and maintenance effort.

3. Solutions

Now that we know the problems we face - what can we do about them? The techniques described below tackle each problem in turn.

3.1 Design For Testability

The first problem is the inappropriateness of isolation testing for C++ systems. Remember, this is due to an increase in the number of stubs required (making isolation more expensive), and problems with stubbing constructors (making isolation impossible without breaking encapsulation). The increase in the number of dependencies is in many cases an unavoidable result of OO design. However, consideration of dependencies during design can help avoid an unnecessary increase in the dependencies between components – thus reducing testing effort (see [Lakos] for more details).

To solve the isolation problem, consider what sorts of classes *could* be stubbed: classes with no (private) data members. Abstract Base Classes (ABCs) are a perfect example of stubbable classes.

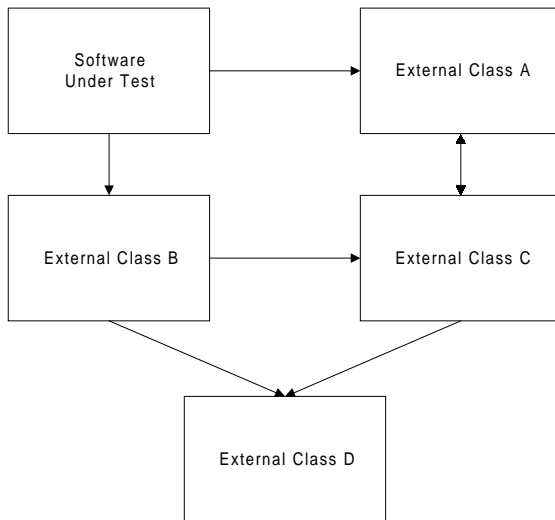
ABCs and their partners, Concrete Implementation Classes (CICs) form a technique for completely separating the class interface from its implementation. Normally, a C++ class declaration defines both the interface (the public part) and some features of the implementation (the private part) in a single place. As we have seen, this causes problems when we attempt to stub these classes. In the ABC/CIC technique, the class is split into two classes:

1. the ABC which defines the (public) interface to the class (as pure virtual member functions);
2. the CIC which inherits (publicly) from the ABC, and provides the implementation of the class⁶.

⁶ By implementing (overriding) the pure virtual functions declared in the ABC.

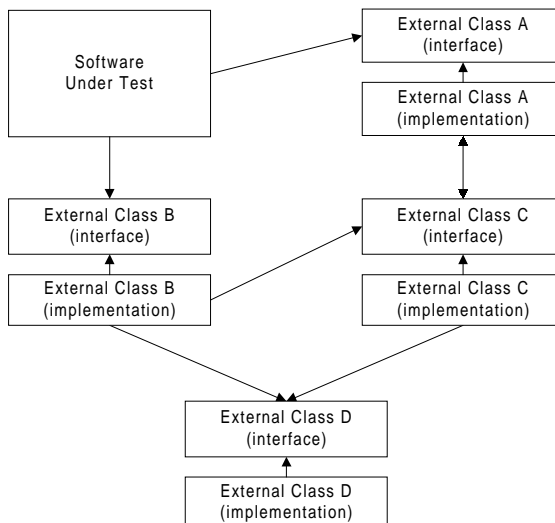
Clients of the class depend only on the abstract base class, not on the implementation class. To stub the class, we retain the ABC, but provide an alternative (stub) implementation class.

Consider the following small sub-system (arrows indicate a dependency):

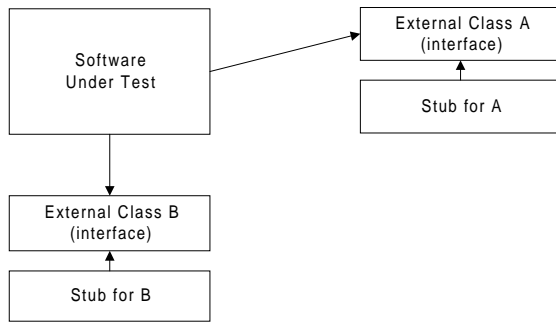


Attempting to isolate the software under test would involve stubbing all the external classes – which we have seen is difficult or impossible.

Now consider what happens if we use ABCs to separate the interface and implementation for each of the external classes:



On the surface, this design looks *more* complex – after all there are more classes! But consider how we would test the class with this design. We can stub the external classes A and B, and external classes C and D can then be omitted from the test altogether (they are only depended on by the implementations of A and B, which have been stubbed):



Note that in this example we have omitted the details of how objects are created. Clearly, the code which *creates* the object will depend on the implementation class (whereas code which only *uses* the object depends only on the interface class). The usual technique for managing these dependencies is to apply the “Factory Method” pattern.

This technique looks to be the answer to our isolation testing problems.⁷ Unfortunately, it is not.

There is an overhead in the use of the virtual member functions – each virtual member function call involves an additional indirect memory access⁸. This overhead would be unacceptable in many systems, if applied throughout. There is a trade-off to be made between the goals of efficiency, testability and maintainability.⁹

In addition, as testers we are often not able to influence the design of the system. The testing phase is considered by some to start after design is complete, rather than as an essential part of the complete software development process.

In the real world, the most we can hope for is that ABCs are used to isolate significant sub-systems. Within those subsystems, though, we shall still be restricted to bottom-up integration testing techniques.

3.2 Instrumentation for Testing

We have resigned ourselves to using bottom-up integration testing, for at least some of our class (unit-level) testing. Some sub-systems will be stubbed, but we will be integrating with a number of (already tested) units.

It is difficult to perform unit-level testing during an integration test, for the reasons described above. Since we will be concentrating much more on integration testing, we want to make it as painless as possible.

The problems we expect during bottom-up testing are due to the data and behaviour of the software under test being hidden from the test harness – either by the data hiding features of the language, or by the processing performed by the other units which are linked with the test.

⁷ It also has re-use and maintenance benefits: new implementation classes can be added to the system without changing client code. See [Martin] for more on this.

⁸ Virtual function calls have also been shown to reduce the effectiveness of modern super-pipelined CPUs, further impacting performance.

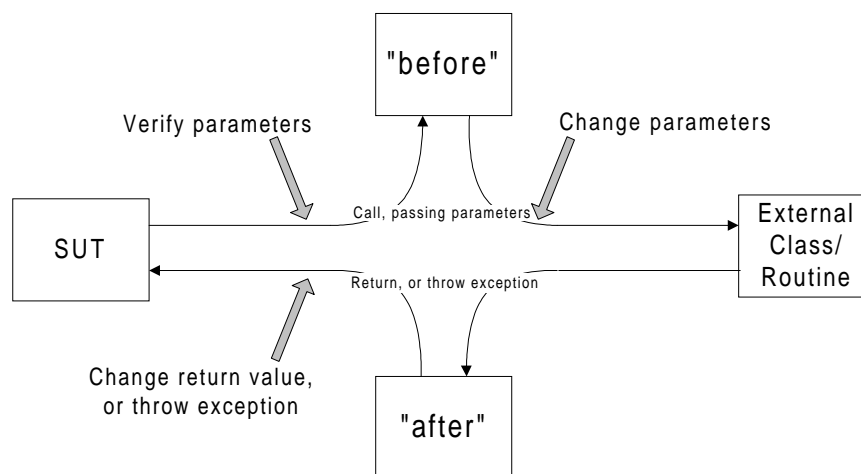
⁹ An alternative solution is to hide implementation data behind a `void* pointer_to_impl_data` (rather than behind an abstract class interface). This approach suffers from the same efficiency and overhead problems as the use of ABCs.

The first stage towards making integration testing easier is to allow the test harness access to the implementation details of the class under test, through a `friend` declaration. This allows the test harness to:

1. verify the correct implementation of the class by examination of its private data;
2. call private (helper) functions of the class to ensure that they are fully tested;
3. initialise private data of the class to force particular execution paths.

The second stage is to allow the test harness access to the calls made from the class under test to the other linked-in units. This is achieved through “call interface instrumentation”, an extension to the commonly used source code instrumentation technique used for coverage analysis.

Call-back functions are called immediately **before** and **after** the original call; the call-backs “wrap” the original call:



Wrapping gives the test harness access to:

1. the order that calls are made;
2. the parameters passed to each call;
3. the return value from the linked-in function.

In addition, the test harness is able (through the call-back mechanism) to:

1. modify the return value passed back to the software under test;
2. throw an exception instead of returning a value;
3. modify any “output” parameters that are passed back to the software under test;
4. modify any other parameters before they are passed to the linked-in function.

Wrapping makes it easy to force “special cases”, such as error conditions, in order to test all parts of the class under test. For example, to verify that the software under test correctly handles the possibility of an exception thrown by a 3rd party database access library might be impossible using normal bottom-up testing techniques – how would we force the library to throw the exception?

Using wrapping, such as test is simple to implement. The appropriate call to the library is wrapped, and the call-back functions cause an exception to be thrown. This exception is then propagated to the software under test, as if it came directly from the database library.

Call interface instrumentation is in many ways similar to coverage instrumentation. Both techniques are implemented using source code instrumentation, whereby instrumentation code is added to the source code to provide the test harness with information about the execution of the software under test which would normally be hidden.

Coverage instrumentation provides the test harness with information on the proportion of functions, statements, decisions and conditions which have been exercised during the test. The instrumentation code records the execution of each statement (or decision, condition etc.) as it happens.

Call interface instrumentation provides the test harness with information on the exact calls which have been made, the order in which they occurred, and the parameters passed to each call. The instrumentation code records the calls made, and the parameters passed.

Wrapping, in combination with a bottom-up test strategy, provides many of the benefits of isolation testing, with increased flexibility, and without the scalability and maintenance problems which are associated with stubbing C++ classes. The following table compares the features of isolation testing and wrapping:

| | Isolation Testing | Wrapping |
|--|-------------------|--------------|
| Check call order | ✓ ⁱ | ✓ (optional) |
| Check parameters | ✓ (optional) | ✓ (optional) |
| Call original function | ✗ ⁱⁱ | ✓ |
| Set return value | ✓ | ✓ (optional) |
| Throw exception | ✓ (optional) | ✓ (optional) |
| Change output parameters | ✓ (optional) | ✓ (optional) |
| Call original function with modified parameters | ✗ | ✓ |
| Use with system calls | ✗ ⁱⁱⁱ | ✓ |
| Use selectively (based on call-site, as well as function called) | ✗ | ✓ |
| Original function is linked with test | ✗ | ✓ |

ⁱ Because a stub must always provide a return value, and cannot call the original function, it must always know “where we are in the test” in order to create the correct return value.

ⁱⁱ A stub function is simply a replacement implementation for the original function. The two cannot be linked together in the same test harness, since they have exactly the same linkage name.

ⁱⁱⁱ In general, isolation testing cannot be used with system calls. Consider a test harness which stubbed the `exit()` function: how would it terminate?

3.3 Re-use of test cases

Whenever we re-use a software component in a context which requires re-testing, we should also re-use the corresponding test cases. This re-use will be done not just at the test planning level, but through direct re-use of the test case code. Designing and implementing re-usable test cases will require more effort to make them re-usable, in the same way that creating re-usable components is harder than creating single-purpose components. In both cases, the pay-off for the initial investment comes later, when the component is re-used, either elsewhere in the system, or in other systems.

3.3.1 When to Re-use?

The usual example of re-use in C++ is through the inheritance mechanism. In particular, the use of public inheritance implies the existence of an “isA” relationship between the classes. Of course, the compiler cannot enforce the rule that public inheritance is used only when there is an “isA” relationship – the compiler has no knowledge of the semantics of the classes. However, using public inheritance when there is no such relationship is asking for trouble. Any function which is declared with a Base& (or const Base&) parameter may in fact be passed a reference to a Derived object. If Derived “is-not-A” Base, then the function is likely to behave unexpectedly (i.e. wrongly).

By re-using the base class test cases when testing the derived class(es), we can verify that the “isA” relationship holds true, and that the derived class correctly implements the interface defined by the base class. This approach can even be used for abstract base classes: test cases can be written which verify the correct implementation of the interface provided by the abstract base.

The above applies equally to re-use through template instantiation. Each template makes certain assumptions about the types on which it will be instantiated: that they are copyable, assignable, have an equality operation etc. These assumptions are both syntactic (`obj1 == obj2` is legal) and semantic (`==` defines an equivalence relation). Unfortunately, there is no way to make these assumptions explicit in the C++ language¹⁰. To verify that the assumptions have not been violated, write a set of re-usable test cases for the template, and use those test cases to test each instantiation.

Re-usable test cases will in general be behavioural tests, based on the externally visible functionality of the class under test. It is also possible to re-use structural test cases, where the code in question has not been replaced (i.e. for inherited members, or templates for which no specialisation has been defined).

3.3.2 Factory Classes

Writing a re-usable test case is, in most respects the same as writing a normal test case. Extra care is needed to ensure that no unwarranted assumptions are made about the types of objects.

For inheritance re-use, the test script must use references or pointers instead of simple objects. For template re-use, the test script itself is a template, parameterised by the same types as the template under test.

Most test cases create one or more objects, in preparation for the test proper. How can a re-usable test script create objects, without knowing the type of the objects being tested. The solution, again, is to use the Factory Method pattern.

The test case is passed a Factory object as a parameter, and uses the factory to create objects as needed. A different Factory class is written for each class which is to be tested. Whenever the test case is used to test a class, it is passed the corresponding Factory object.¹¹

To ensure that different Factory objects can be substituted as necessary, the Factory classes form an inheritance hierarchy of their own, which precisely mirrors that of the classes to be

¹⁰ c.f. the constrained genericity and design-by-contract ideas described in [Meyer] and supported by the Eiffel language.

¹¹ The approach described here is of the more general form, where the test and factory classes are distinct. In many common cases (including most unit-level tests) the test class creates objects of only one type. In such cases the test and factory classes can be combined into a single class.

tested. This parallel inheritance hierarchy means that if a `Circle` “isA” `Shape`, then a `CircleFactory` “isA” `ShapeFactory`.

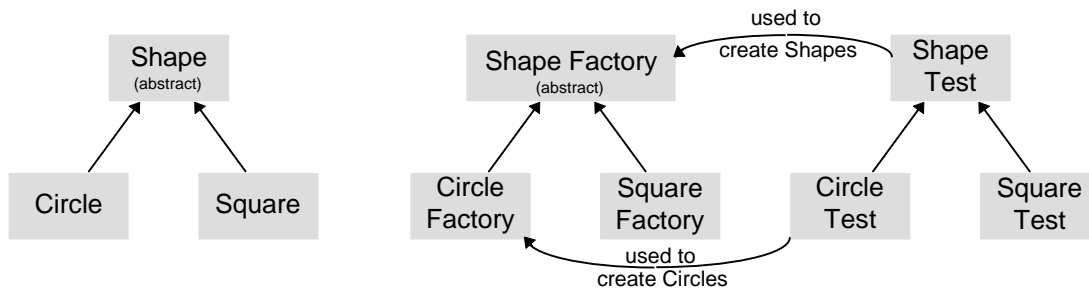
In the following diagram, `Shape` is an abstract class, and `Circle` is derived (publicly) from `Shape`. `ShapeTest` contains the test cases which test for “Shape-ness”. `CircleTest` contains the test cases which test for “Circle-ness”. Both consist of re-usable test cases, and `CircleTest` automatically runs the `ShapeTest` test cases on `Circles`, to ensure that `Circles` are indeed `Shapes`.

`ShapeTest` needs to create `Shapes` before it can test them; it uses a `ShapeFactory` (passed as a reference parameter) to do this. Similarly, `CircleTest` needs to create `Circles`, for which it uses a `CircleFactory`.

`ShapeFactory` is an abstract class, since it can’t actually create any `Shapes`. The purpose of `ShapeFactory` is to define an interface for shape-creation which the derived factory classes (like `CircleFactory`) must implement.

`CircleFactory` implements the interface by creating `Circles` (remember that a `Circle` isA `Shape`, so this fulfils the `ShapeFactory` requirements). A hypothetical `SquareFactory` would implement the same interface by creating `Squares`. `CircleFactory` also defines an additional interface specifically for creating `Circles`.

When `CircleTest` runs the `ShapeTest` test cases, it passes it the `CircleFactory` object. This is used by the `ShapeTest` test cases, which test that the `Circles` so created are valid `Shapes`.



In this section we have seen how to use re-usable test cases to keep testing effort for re-usable components within reasonable bounds. But how do we ensure that a component has been tested in all necessary contexts? This question, of *coverage*, is addressed in the next section.

3.4 Inheritance Context Coverage

Whenever a member function is inherited, it can be re-used in a new context – acting on a Derived class object instead of a Base class object. As discussed above, we need to re-test such member functions in each new context (preferably using re-usable test cases).

To ensure that the member function has been thoroughly tested in each context, it is necessary to extend the definition of existing structural coverage metrics to take into account the *inheritance context*. For each metric, coverage is measured separately for each context. The current context is the (dynamic) type of the object on which the member function is acting.

The traditional coverage metrics can be calculated from the inheritance context coverage data by aggregating the coverage achieved across all contexts.

When calculated for a specific inheritance context, function entry-point coverage provides a basic verification that each function has been calculated in the specified context. Typical recommended coverage requirements are:

1. once-full context coverage: 100% (statement, decision) coverage in at least one context, 100% entry-point coverage in all contexts
2. strict context coverage: 100% (statement, decision) coverage in all contexts

Use of inheritance context coverage metrics can be used to ensure that re-used code has been tested in all necessary contexts.

3.5 State Coverage

When testing a class whose behaviour is modelled by a state machine, we will naturally use our knowledge of the possible states, and the transitions between them, to help design suitable test cases.¹² To ensure that we have not missed any cases, enhanced coverage metrics can be used to measure coverage separately for each possible state.

For example, consider a `bounded_stack` class. It will have states `empty`, `normal` and `full`. To ensure that the class's behaviour in each state has been tested, coverage is measured separately for each state.

The user must provide a function which determines the “current state”. In all other respects, state coverage is very similar to inheritance context coverage: coverage is measured separately for each state/context, and coverage metrics can be calculated for a specific context, or across all contexts.

Typical recommended coverage requirements are:

1. once-full state coverage: 100% (statement, decision) coverage in at least one state, 100% entry-point coverage in all states
2. strict state coverage: 100% (statement, decision) coverage in all states

Astute readers will have realised that achieving strict state coverage will often be infeasible. If the class is to behave differently in each state, then there must be some code which is executed differently, depending on the state. For example, in the `bounded_stack` class, the `push()` member function is likely to include a decision of the form `if (empty())`. Clearly, this decision will only branch true in the `empty` state, and will only branch false in the `normal` or `full` states.

A structural coverage metric for which 100% coverage is infeasible is almost useless as a measure of testedness since it is impossible to know what “75% coverage” means: “the 3 feasible cases have all been covered” or “9 of the 11 feasible cases have been covered”.

To counter this problem, the enhanced coverage metrics are further modified to allow the tester to define particular coverage items (e.g. decision branches) as being infeasible in a particular state. Coverage metrics can then be calculated based on the feasible cases only: in the example above the results would be “100%” and “82%” coverage respectively – making the difference clear.¹³

In the inheritance context case, achieving full coverage usually is feasible. Inherited member functions should behave (mostly) the same when applied to a derived class as they do in the

¹² See [Binder] for a description of one such approach.

¹³ To ensure that “cheating” is not possible, the raw coverage metric is also reported.

base class; any differences should be encapsulated as virtual member functions which are overridden. Thus, most member functions do not contain decisions based on the actual type of the underlying object. However, in those rare cases (involving the `dynamic_cast` operator) where there is a problem, the same approach can be applied.

3.6 Object-Oriented Complexity Metrics

A number of Object-Oriented static complexity metrics have been proposed, including those described in [Chidamber], [Abreu], [Martin] and [Bansiya]. The proposed metrics attempt to measure the important features specific to object-oriented software systems. These include the use of encapsulation and data hiding, the use of polymorphism, the use of abstract interfaces and the “size” of the system in terms of its classes and interactions between them.

Although some validation work has been done on these metrics, further investigation is required to identify those which are most useful as indicators of defect density and coding, testing and maintenance effort. Of course, for the necessary, large-scale, validation studies to be feasible a tool to automate the collection of the metrics under consideration must be available. In the absence of definitive validation results, these metrics are still useful as an aid to practitioners’ engineering intuition in determining the likely defect and effort “hot-spots”.

4. Summary

We have seen that testing C++ software presents a number of new challenges. Increased numbers of dependencies, combined with the exposition of class implementation details in header files, make isolation testing infeasible in most cases. The resulting switch to bottom-up testing forces us to perform unit-level testing on larger and larger integrations of units. The re-use of components, through inheritance and instantiation, forces us to consider the re-use of test cases. New coverage and complexity metrics are required to maintain the value of structural coverage analysis for test case design.

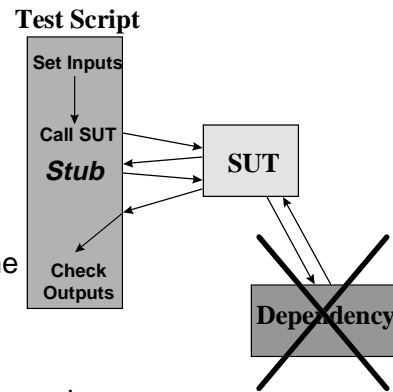
This paper has presented solutions to all these problems. The use of tools implementing and supporting these solutions, such as IPL’s *Cantata++ V2* [IPL], will result in more efficient and more effective C++ testing.

5. References

- [Abreu] Abreu F B and Melo W (1996). "Evaluating the impact of object-oriented design on software quality", in Proc. 4th International Software Metrics symposium, IEEE Computer Society Press, 1996 (available at <http://www.cs.umd.edu/users/melo/papers/CS-TR-3536.html>).
- [Bansiya] Bansiya J and Davis C (1997). *An Object-Oriented Design Quality Assessment Model*, IEEE Transactions of Software Engineering, August 1997 (available at <http://indus.cs.uah.edu/research/qmood++/bansiya.htm>).
- [Binder] Binder R. *The FREE Approach to Testing Object-Oriented Software*, (<http://www.rbsc.com/pages/FREE.html>).
- [Chidamber] Chidamber S R and Kemerer C F (1991). *Towards a metrics suite for object-oriented design*, in Proc. 6th OOPSLA Conference, ACM 1991, pp.197-211.
- [Harrold] Harrold M J, McGregor J D and Fitzpatrick K J (1992). *Incremental Testing of Object-Oriented Class Structures*, Proceedings of the Fourteenth International Conference on Software Engineering, 1992, pp. 68 - 80.
- [IPL] Cantata++ product information at <http://www.iplbath.com>.
- [Lakos] Lakos J (1996). *Large Scale C++ Software Design*, C++ Report, June 1996 onwards.
- [Martin] Martin R C (1996). *The Dependency Inversion Principle*, C++ Report, May 1996 (also available at <http://www.oma.com>).
- [Meyer] Meyer B (1997). *Object-Oriented Software Construction (2^{ed})*, Prentice Hall, ISBN 0-13-629155-4.

Traditional Dynamic Testing

- Test software by running it
 - Unit testing and Integration testing
 - Set inputs; call SUT; check outputs
 - Automated scripts for repeatability
 - Re-run tests when software changes
- Test as soon as possible
 - Find faults earlier (cheaper)
 - Test before dependencies are done
- Provide a “stub”
 - Stub is part of the test
 - Stub has same interface as real dependency
 - Stub provides test-specific implementation
 - + same header (.h) file but different code (.c) file



Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL

1

Traditional Code/Test Analysis

- Have we tested everything?
 - Every function/source line/statement/decision/boolean?
 - Use Structural Coverage metrics
 - + collect data during dynamic test run
 - + identify areas not covered and add test cases to test them
- Too expensive to test everything?
 - Calculate Static Code Size and Complexity metrics
 - Estimate test effort
 - + number of decisions; number of calls
 - Estimate defect density
 - + lines of code (!); number of decisions
 - Aim to maximise benefit for minimum cost
- Avoid dangerous constructs; keep complexity low

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL

2

C++ Problems

- Data hiding prevents access to private data
 - Difficult to force desired behaviour
 - + set private data or call private methods directly
 - Difficult to check result state
- Want to use isolation testing during unit testing
 - OOD => many small, interacting components
 - Interfaces involve additional objects (parameters/returns)
 - Must stub all methods of a class, not just one
 - C++ header (.h) is not just the interface
 - + also contains private data declarations
 - + which might be class types with their own private data
 - Even more stubs to write!
 - Isolation Testing is more difficult than with C

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 3

More C++ Problems

- Ever tried writing a stub for a constructor?
 - Keyboard has a (private) Port member
 - Port has no default constructor

```
class Port {                class Keyboard {
public:                      ...
    Port(int portnum);      private:
    ...                     Port port;
};                            };
```

- We're testing a client of Keyboard so we want to stub it
- Stub for Keyboard::Keyboard() must initialise port

```
// test_client.cc
...
Keyboard::Keyboard() : port(STUB_KEYB_PORTNUM){ ... }
```

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 4

Why Is This Bad?

- The stub is part of the test for the client
 - The client doesn't care about Ports
 - The test shouldn't have to either
 - Encapsulation has been broken
- This will apply to every client of Keyboard
 - Maintenance nightmare!
 - What if Keyboard changes to use FasterPort ...
 - Every test for every client will need to be updated
- Encapsulation aims to protect from changes
- Isolation testing aims to protect from changes
- Both have failed

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 5

C++ Forces Integration Testing

- Isolation Testing is
 - Always difficult
 - + hard work writing tests through public interface only
 - Often expensive
 - + sheer number of stubs is daunting
 - Sometimes impossible
 - + stubbing constructors causes more problems than it solves
- No isolation => bottom-up integration test
 - Need to test for both unit and integration faults
 - Difficult to force desired behaviour
 - + the rest of the system “gets in the way” especially for unit-level testing
 - + can't use stubs to “inject” data to force behaviour
 - Information hiding still prevents access to private data

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 6

C++ Problems: Re-use of Code

- OOD “marketing” promotes re-use
- Re-use of code => effort saved
 - Extra design effort initially
 - Save effort every time we re-use
- M.J.Harrold claimed re-test necessary when re-used in new context
 - e.g. inherited method used in derived class or instantiation of a template
 - Any virtual methods may have changed semantics
- Re-test *without* re-use of test cases => extra effort
 - Test effort might be 20%-50% of development
 - Re-test effort negates savings from code re-use

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL

7

Code Coverage and Complexity

- Achieving 100% decision coverage is (too) easy
 - + individual methods are small
 - + design complexity is in object interactions/polymorphic dispatch
 - + common metrics don't even attempt to measure how well this complexity has been tested
- Defect estimation
 - + bugs more likely in interactions between classes
 - + lines of code probably still a reasonable guess!
- Test effort estimation
 - + Test effort depends on interactions between classes
 - + Methods score low on (traditional) complexity
 - + Polymorphic calls more important than decisions
- New metrics are required

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL

8

Test Classes As Friends

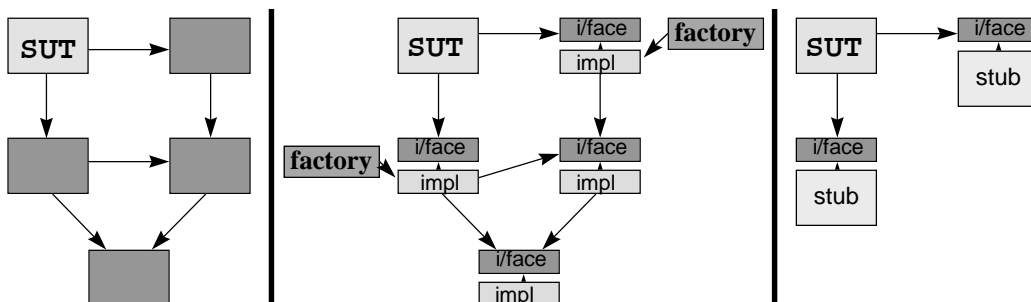
- Each class in the SUT has a corresponding test class
- Make the test class a friend of the SUT class
 - The test class has access to private data and methods
- Allows us to write White-Box test cases
 - Call private methods directly
 - + easier to test them thoroughly (achieve coverage)
 - Set private data directly
 - + easier to test boundary cases
 - Check private data directly
 - + more likely to find obscure implementation faults
 - + e.g. deep versus shallow copy constructor
 - or other pointer corruption
 - + need complicated test case to find the fault in a black-box test

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 9

(Re-) Design For Testability

- Separate interface and implementation classes
 - Clients use *interface*; use factories to create *implementations*
 - Test can use a separate “stub” implementation class
 - Test size and complexity reduced
 - More flexible design but slower and more source code



Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

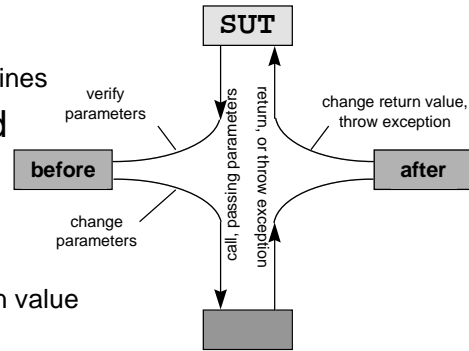
IPL 10

Call Interface Instrumentation

- We're doing bottom-up testing
 - Trying to achieve unit-level testing during integration test
 - Need to give test script access to implementation
 - + private data and methods
 - + calls made within the SUT
 - + calls to external classes/routines

- Call-backs made **before** and **after** the original call

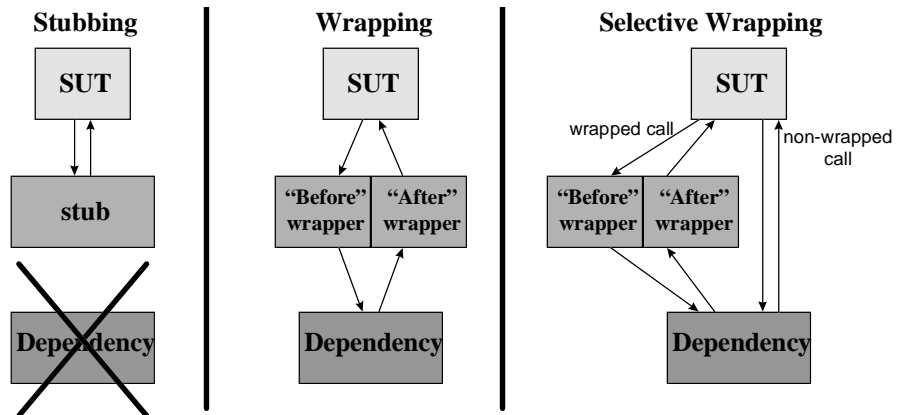
- Call-backs “wrap” original call
- Wrapping gives access to
 - + call order; parameters; return value
- Allows test script to
 - + check calls made; change return or parameters; throw exception



Testing C++: Why It's Hard And How To Do It Well
 © 1999 IPL Information Processing Ltd

Comparison With Stubbing

- Wrapping occurs around (selected) normal calls
- Stubbing completely replaces (all) normal calls
- Stub and wrapper both considered part of test script



Testing C++: Why It's Hard And How To Do It Well
 © 1999 IPL Information Processing Ltd

Uses Of Wrapping

- Force error conditions for specific test cases
 - e.g device errors, out-of-memory, file not found
 - + wrap selected calls; modify wrappers to force error condition
- Achieve effective isolation without stubbing
 - Don't want "real" called functions to be invoked
 - Want to force specific paths
 - Modify generated wrappers to REPLACE original function
 - + check call order; check parameters; set return value
- Record calls made
 - Debugging/tracing/profiling or verify UML sequence diagrams
 - + wrap important calls
 - + modify generated wrapper function to log results
- All possible without changing project source code

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 13

Re-use of Test Cases

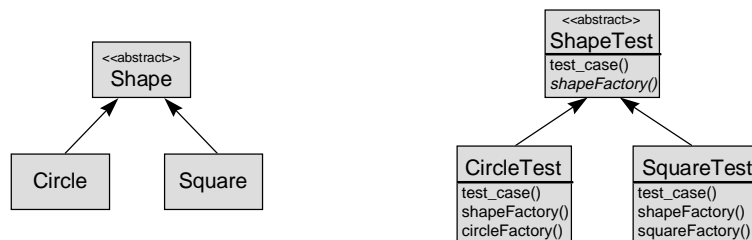
- Re-using test cases verifies that a derived class has the same semantics as its base
 - public inheritance means "isA"
 - re-using test cases verifies that the "isA" holds true
- For methods which are overridden
 - re-use black-box test cases
 - can write BB test cases for abstract classes, just for re-use
 - also applies to templates which are specialised
- For methods which are inherited
 - re-use black- and white-box test cases
 - also applies to templates which are instantiated

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 14

Parallel Inheritance Hierarchies

- Test for derived class inherits from test for base class
 - Derived test class calls base test class's test cases
 - Write tests even for abstract classes so they can be re-used
 - SUT classes and test classes form parallel inheritance hierarchies
 - Use a "Factory Method" to create objects for the test
 - Factory method for abstract class is a pure virtual method



Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 15

Object-Oriented Coverage

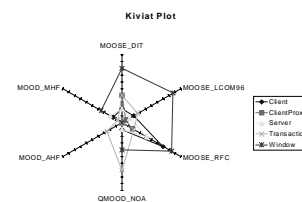
- A method can be called in more than one context
 - On *base* class or *derived* class objects
 - + behaviour can change because of virtual members
 - Many classes are inherently state-based
 - + behaviour is different in each state
 - + e.g. `bounded_stack` class has states `empty`, `normal` and `full`.
- Measure coverage separately in each context
 - For each function, statement, decision, expression
 - Contexts correspond to derived classes or to states
 - Typical coverage requirements
 - + strict: 100% statements, decisions in all contexts
 - + once-full: 100% statements, decisions in at least one context, all functions entered in all contexts
 - + minimal: 100% statements, decisions (sum across all contexts)

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 16

Object-Oriented Complexity

- Dependencies between classes
 - + Robert Martin's Dependency metrics
 - + MOOSE ("Sloan School")
 - + MOOD
 - + QMOOD
- Use of public/protected/private virtual/static methods and data
 - + MOOD
 - + QMOOD
 - + lots more raw counts
- Complexity of individual methods
 - + cyclomatic complexity, Halstead
 - + line and statement counts



Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 17

How Much Testing?

- Estimation
 - How long to isolation test? (Cyc.Complexity, RFC, MHF)
 - How long to integration test? (CBO)
- Where should we concentrate our testing?
 - More complex code is more likely to be wrong (RFC)
 - Bigger code is more likely to contain defects (LOC, #STMTS)
 - Code which is re-used is more important to get right (NOD)
 - What makes an object-oriented design good?
 - + minimisation of dependencies (AHF)
 - + separation of interfaces (Martin metrics, LCOM, CAM)
 - + maintainability (DIT, NOC, Hierarchy Quality)
 - Combine with coverage data from previous testing
 - + identify "hot-spots" with high complexity and low coverage

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 18

Cantata++ v2

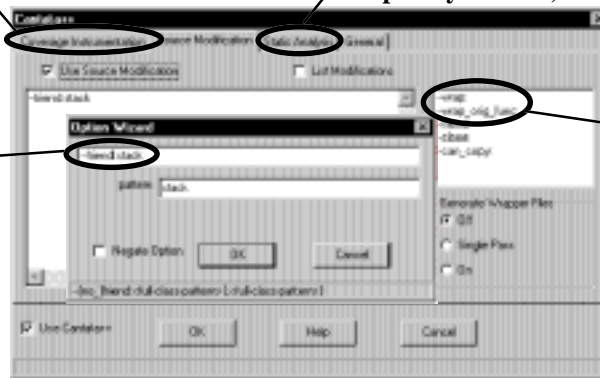
- IPL's ***Cantata++ v2*** supports all these solutions
 - see www.iplbath.com for more information

Coverage
(OO and
traditional
metrics)

Static Analysis
(OO and traditional
complexity metrics)

Stubbing
(isolation
testing)

Friends
(white-box
testing)



Wrapping
(easier
integration
testing)

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 19

Conclusion

- Testing C++ presents many challenges
- Solutions
 - Friends
 - Design for Testability
 - Instrumentation for Testing
 - Re-use of Test Cases
 - Enhanced (context-specific) Coverage
 - Object-Oriented Complexity Metrics
 - IPL's ***Cantata++ v2***

Testing C++: Why It's Hard And How To Do It Well
© 1999 IPL Information Processing Ltd

IPL 20

TESPRA

A practical and integrated method for Test Control, Reporting and Estimation

Johan Swinnen

Gitek nv - interaction through software

St. Pietersvliet 3, B-2000, Antwerpen, Belgium

Tel: +32 3 231 12 90 - Fax: +32 3 226 10 83

E-mail: JS@gitek .be

1 Introduction

1.1 General

This paper is about TESPRA, a practical and integrated method for Test Control, Reporting and Estimation. TESPRA was originally developed by Corn le Koning, IQUIP Informatica B.V., The Netherlands.

This method is used during (test) projects at Janssen Research Foundation (JRF), a division of Janssen Pharmaceutica N.V. (a Johnson & Johnson Company), Beerse, Belgium. For JRF, TESPRA has been extended by Johan Swinnen, GiTek n.v., Antwerp, Belgium.

The first part of the paper will give an overview of TESPRA. The second part shows an example of TESPRA at JRF.

1.2 Janssen Research Foundation (JRF)

Janssen Research Foundation is a division of Janssen Pharmaceutica, which was founded in 1953. It has 5 products on the WHO list of Essential Drugs (~200). Janssen Pharmaceutica currently produces over 80 pharmaceutical products.

Since 1961 JRF belongs, as a pivot pharmaceutical research company, to Johnson & Johnson. J&J stands for 188 companies in 52 countries with more than 93.000 employees and is active in the consumer, professional and pharmaceutical sector. Of the annual sales, approximately 10% is dedicated to Research and Development. In Belgium the research department is grouped in the Janssen Research Foundation.

Gitek started working in projects at JRF in 1998. The first project was ENDO, an application for registration and analysis of blood samples. Secondly, there was ISIS, where we tested a registration application for molecular structures. We just finished CAVAS, a system that is used in the laboratories during experiments on Beagle Dogs. Due to the results of ENDO and ISIS, and based on FPA¹, TPA² and TESPRA, CAVAS was a fixed price / fixed date project.

At this moment Gitek is involved in two projects:

- DSM, a management and retrieval application for DNA samples. This is also a fixed price project.
- A-Lims: Advanced Laboratory Information Management System.

¹ FPA = Function Point Analysis

² TPA[®] = Test Point Analysis

2 Purpose

The main goals of TESPRA are:

- gaining control over your test project;
- giving detailed and useful reports on the test project;
- making experience-based estimations on future test projects.

If you want information about a test project, you need to do measurements. Keeping measurements costs time and money.

With this in mind, TESPRA was developed with the following characteristics:

- a limited set of metrics;
- specified information and detailed reporting;
- not time consuming (80/20-rule).

3 When to use TESPRA?

TESPRA is developed for test projects and is a method to be used during test projects where you have full responsibility for the test activities.

TESPRA can give you information at all levels of the project, from test engineer (e.g. defects) to test manager (e.g. estimation and progress).

4 What is TESPRA?

TESPRA supports Test Management in controlling the test project, reporting about the test project and estimating future projects.

To obtain these goals TESPRA is based on 4 pillars:

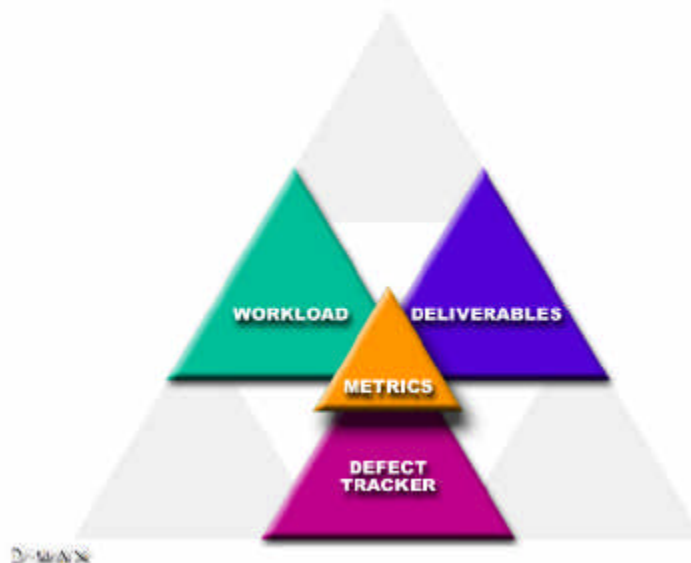


Figure 1

As figure 1 shows, we need to do measurements of workload, defects and deliverables. The results of these measurements are combined in the metrics pillar.

In the next chapters, each pillar will be explained. The pillars are developed in such a way that they can be used on their own. However, the metrics pillars requires information from the other pillars.

4.1 Workload

The test manager as well as the project manager wants to know the status of the hours spent on the project. The test manager wants to control the test budget, the project manager wants to have an overview of the entire project, and therefore needs information about the test project.

In order to get this information, each test team member has to keep a detailed list of the hours spent during the test project.

The more detailed information you gather, the better an analysis can be made. Of course, the characteristics of the project determine the level of detail.

TESPRA requires at least the following items:

- Name;
- Project;
- Date;
- Activity;
- Number of hours;
- Description;
- Test phase.

The pillar workload is used for monitoring and controlling the test process. For instance : every time you gather the information from the test team members, you can determine the project progress and compare it with the budget.

For reporting you can plot graphics and create reports of this progress. At JRF we use the test management approach TMap® which is the standard used by IQUIP Informatica B.V. and Gitek n.v. in the Benelux. See figure 2.

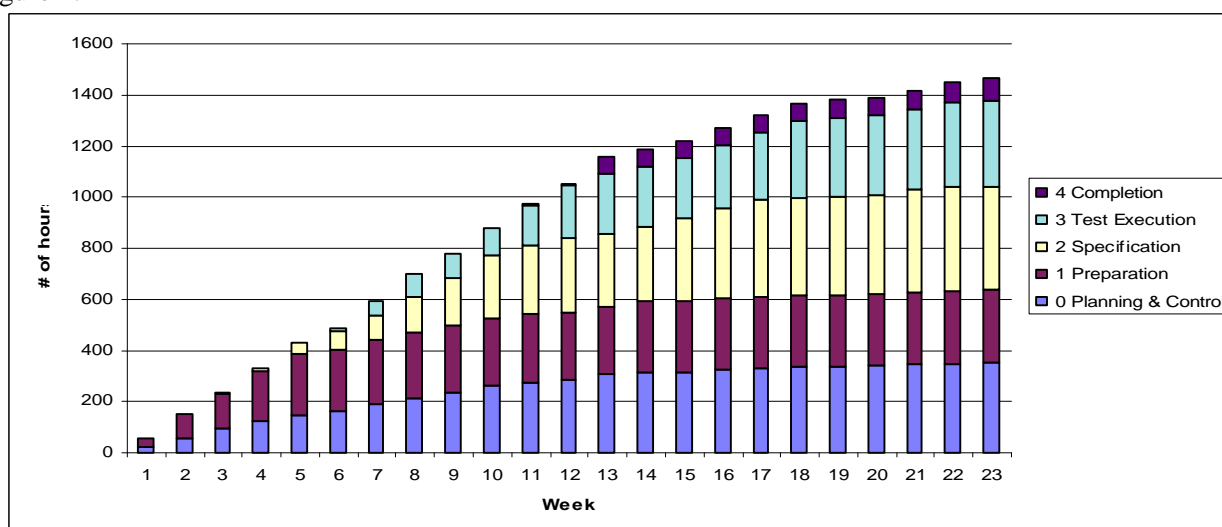


Figure 2: cumulative hours

An important remark is that recording the workload requires discipline. You can't remember in detail what you've done two weeks ago. Moreover, if you have the discipline to record it daily, that makes it cost little time.

4.2 Defect Tracker

In the first place defect tracking is meant to monitor and control the defects.

The following items are required for TESPRA:

- ID;
- Date;
- Finder;
- Severity;
- Cause;
- Status;
- Test script;
- Test case;
- Quality attributes;
- Description.

For reporting you can make a progress report where you can compare status versus severity, severity versus cause and cause versus status. You have an overview of the number of open defects per status and the number of days per status that a defect stays unsolved. See figure 3.

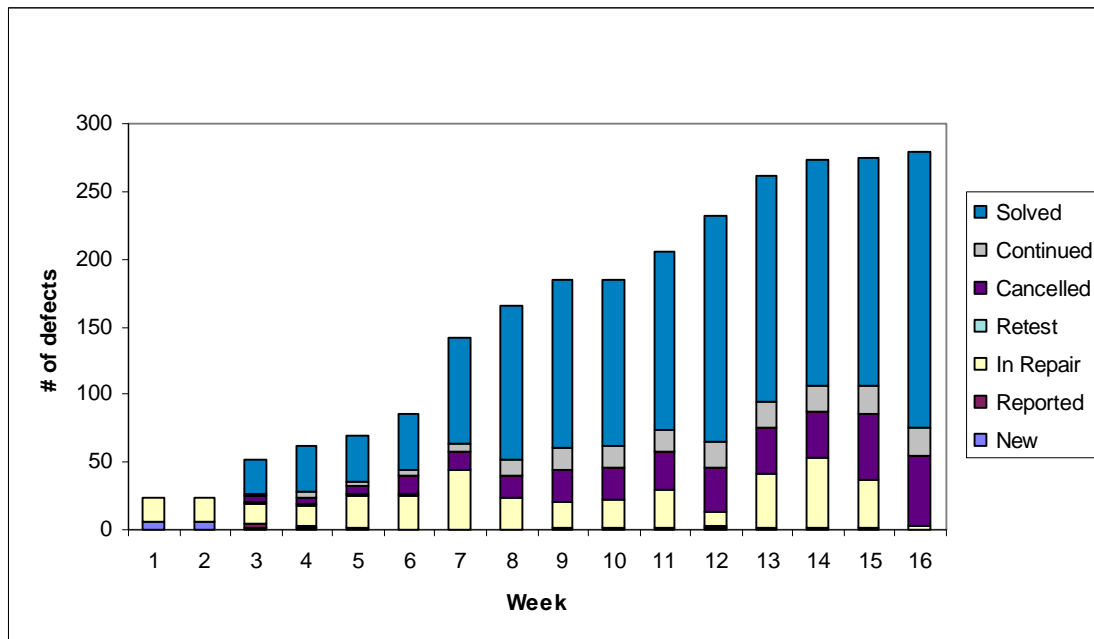


Figure 3: Progress Report on Defects

4.3 Deliverables

As deliverables we consider the formal testware.

The required items are:

- Project;
- Directory;
- Subdirectory;
- Name;
- Extension;
- Source;
- Test phase;
- Size;
- Date of delivery;

Depending on the type of deliverable:

- # of pages;
- # of test cases;
- # of entities;
- # of attributes.

The item “source” states whether the document has to be delivered to the test team (by the developers), or by the test team (to the project leader)..

4.4 Metrics

As figure 1 shows, the three basic pillars can be combined into the fourth pillar.

In the first part of the metrics pillar we gather information from the other pillars, and add other project characteristics, such as name, start and end date. This results in an enormous treasure of information. Because reporting all is ineffective, you have to pick the items to report in relation to the message you want to bring.

The added value of the metrics pillar is the productivity numbers, based on the gathered information. For instance:

- Workload versus Defects gives the number of defects per hour execution.
- Defects versus Deliverables gives the number of defects per test case.
- Workload versus Deliverables gives the number of the hours spent per deliverable.

Of course this list can be extended and adjusted according to the project characteristics.

The second part contains statistical information: averages, minima, maxima and standard deviations.

5 The Advantages

The best measurable advantage is that TESPRA requires little time. As mentioned before, recording workload takes almost no time. Defect tracking is daily practice in every test project.

TESPRA enables the test manager to give well-funded advice and detailed reporting to the project leader so that both are at any time aware of the status of the test project.

The test manager is able to control his test project and the required budget.

The information in the metric pillar is a basis for estimation on future projects, e.g. workload, productivity.

6 TESPRA at Janssen Research Foundation

This paragraph explains how we used TESPRA to estimate the CAVAS project.

6.1 Estimating Workload

As mentioned before we use TMap as the approach for our test projects. We used the information on workload for an estimation as in the next table:

| | ENDO | ISIS | CAVAS |
|----------------------|------|------|------------|
| Planning and control | 36% | 24% | 21% |
| Preparation | 13% | 19% | 23% |
| Specification | 27% | 23% | 29% |
| Execution | 16% | 28% | 22% |
| Completion | 8% | 6% | 5% |

We estimated a higher percentage for the preparation and specification phase because of the difficulty of the subject.

According to the FPA and TPA count, 1400 hours were required for the test project.

6.2 Estimating Productivity

| | ENDO | ISIS | CAVAS |
|-------------------------------------|------|------|-------------|
| # defects / hour execution | 2.12 | 0.63 | 1.3 |
| # defects / test case | 0.24 | 0.49 | 0.44 |
| # hours specification / deliverable | 1.7 | 1.8 | 2.2 |

Also due to the difficulty of the subject, we estimated a lower productivity.

7 Conclusion

In this paper I explained that TESPRA is a method that provides specified information, with a small set of metrics at little cost.

At the moment TESPRA is being used in several projects in companies in Belgium.

For more information, please don't hesitate to contact me.



TESPRA

November 4th, 1999

Johan Swinnen
Gitek n.v., Antwerp, Belgium



Contents

- Introduction
 - History
 - Janssen Research Foundation
- TESPRA
 - Purpose
 - Pillars
- Example at Janssen Research Foundation
- Conclusion



Introduction

- Initial setup at ABP, The Netherlands
- Method developed during first projects at JRF

- Janssen Research Foundation (JRF)
 - 2.698 employees worldwide
 - 1.481 in Beerse, Belgium
 - Development of over 80 new compounds

**JANSSEN
RESEARCH
FOUNDATION**



Introduction

- Janssen Pharmaceutica
 - Founded in 1953
 - 17.484 employees
 - 5 products on WHO list essential drugs

- Johnson & Johnson
 - 188 companies
 - 52 countries
 - 93.100 employees



Johnson & Johnson



Definition

TESPRA is a practical and integrated method for:

- Test Control
- Test Reporting
- Test Estimation



Purpose

The purpose of TESPRA is

to provide specified information,
with a small set of metrics,
at little cost.

- Use during Test Projects
- Information for all levels



Pillars



Workload

Required items:

- Name, Project, Date
- Activity
- Number of Hours
- Description
- Test Phase

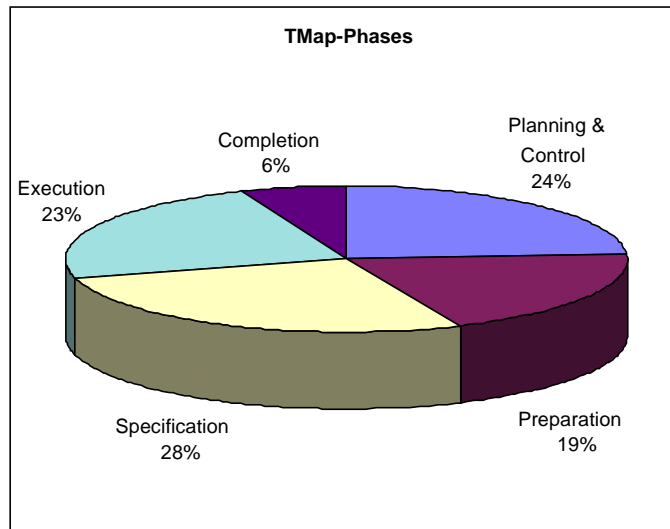


Workload : Example

| | |
|------------------------|-----|
| Test Execution | 16% |
| Test Scripts | 16% |
| Communication | 9% |
| Logical Specification | 6% |
| Physical Specification | 5% |
| Defect Tracking | 5% |
| ... | ... |



Workload : Example Continued





Workload : Example Continued

| Tester | Date | Project | Subproject | Activity | Hours | Description | Tmap-phase |
|--------|----------|---------|------------|--------------|------------|------------------------|-----------------|
| JSW | 29-06-99 | 02 ISIS | 022 RETEST | script | 2.0 | Completion test script | 2 Specification |
| JSW | 29-06-99 | 02 ISIS | 022 RETEST | execution | 1.0 | Execution open defects | 3 Execution |
| JSW | 29-06-99 | 02 ISIS | 022 RETEST | defects | 1.0 | Update defect tracker | 3 Execution |
| JSW | 29-06-99 | 02 ISIS | 022 RETEST | execution | 2.0 | Syntactic Test | 3 Execution |
| JSW | 29-06-99 | 02 ISIS | 022 RETEST | execution | 2.0 | Semantic Test | 3 Execution |
| | | | | Total | 8.0 | | |



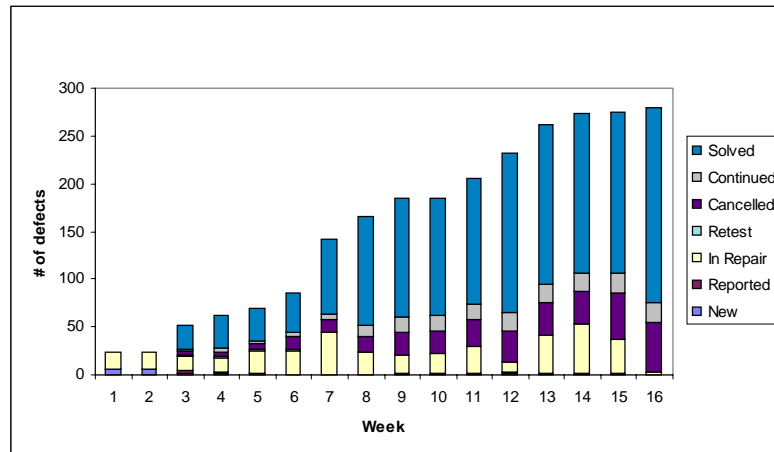
Defect Tracker

Required items:

- ID, Date, Finder
- Severity
- Cause
- Status
- Test Script, Test Case
- Quality Attribute
- Description



Defect Tracker : Example



Deliverables

Required items:

- Project
- Directory
- Name, Extension
- Date of Delivery
- Source
- Test Phase
- Size



Deliverables

Depending on type of deliverable :

- # of pages
- # of testcases
- # of entities
- # of attributes
- # lines of code



Metrics

- Numbers on productivity
 - Workload \Leftrightarrow Defects
 - # defects / hour execution
 - ...
 - Defect \Leftrightarrow Deliverables
 - # defects / testcase
 - ...
 - Workload \Leftrightarrow Deliverables
 - # hours specification / deliverable
 - ...



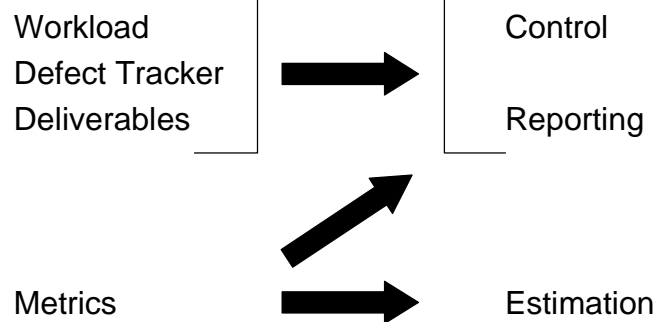


Metrics

- Project Characteristics
- Statistical Information
 - Averages
 - Minima, Maxima
 - Standard Deviations



TESPRA





Estimation at JRF : Workload

| | ENDO | ISIS | CAVAS |
|--------------------|------|------|-------|
| Planning & Control | 36% | 24% | 21% |
| Preparation | 13% | 19% | 23% |
| Specification | 27% | 23% | 29% |
| Execution | 16% | 28% | 22% |
| Completion | 8% | 6% | 5% |



Estimation at JRF : Productivity

| | ENDO | ISIS | CAVAS |
|--|------|------|-------|
| # defects / hour | 2.12 | 0.63 | 1.3 |
| # defects / testcase | 0.24 | 0.49 | 0.44 |
| # hours specification / deliverable | 1.7 | 1.8 | 2.2 |



Conclusion

TESTPRA is a method

that provides specified information,
with a small set of metrics,
at little cost.



Questions ?

Gitek n.v.
interaction through software

Sint-Pietersvliet
B - 2000 Antwerp
Belgium

JS@gitek.be



Extreme Quality: What Can We Learn from the Computer and Video Games Industry?

Thomas A. Drake
Quality Architect, Quality Advocate
Enterprise Management and Information Technology Consulting

Coastal Research & Technology, Inc.

(CRTI)
5063 Beatrice Way
Columbia, Maryland 21044
United States of America

<http://www.coastalresearch.com>

E-mail: tadrake@earthlink.net

Abstract:

What can we learn from the computer and video games industry that is relevant for quality and points to the growing necessity for sustained high quality software development? What does the fascinating and ever exciting world of computer and video games hold out for the future of quality in our increasingly software-centric information technology systems and services? An industry survey and analysis of this interesting and highly dynamic segment of the business provides some perspective on where both testing and quality are heading in a business where “game play” is everything for both the development house and the customer who “plays” the game.

Key words:

Extreme quality, testing, quality assurance, video games, computer games, console games, lessons learned, quality centered development methods, software development, creative software product development and process practice, entertainment technology, software engineering

© Copyright 1999 by Thomas Drake. All Rights Reserved

The Computer and Video Game Industry

The computer and video console game industry is making it big today. In 1998, United States video game revenues grew to some \$6.3 billion dollars. This figure is just short of the 1998 U.S. box office sales for movies of \$6.95 billion dollars. Last year, some 181 million games were sold in the United States. This number equates to almost two games for every household in the United States alone.

Sony has reportedly sold some 60 million units of its Playstation video game console worldwide since its debut in 1995 and is set to introduce its high end Playstation 2 DVD machine next year. The Nintendo 64 has also sold tens of millions of its own video console machines and is developing a new high end “4th generation” DVD system – a 256-bit 400 Mhz IBM PowerPC chip based super console code-named the Dolphin and set to come to market in the year 2000. Sega recently debuted its “3rd generation” Dreamcast console in the United States on 9 September 1999 and took in approximately 98 million dollars in revenue the very first day. Even the portable 8-bit Nintendo Game Boy with its low single digit Mhz plus processor has sold over 80 million units since its 1989 release.

If you think videogames are child’s play think again. The Nintendo 64 game called *Zelda: Ocarina of Time* has grossed nearly \$140 million by itself since its release. Sony’s game division contributed some 40 percent of Sony’s total corporate profits. Some analysts are predicting that computer and videogame revenues for 1999 will top \$8 billion and could actually exceed movie revenues for the first time in history.

Any visit to your local computer or electronic department store will reveal shelf upon shelf of games that one can buy and play for their computer or video game machine. Many of the computer games are tuned for the very latest in 3-D hardware cards and high-speed graphic chips. It is now big business and many, many people now earn their livelihood in this sector of the entertainment industry.

Examining the employment classified sections of the computer game trade journals and related industry publications reveals requirements for software engineers, programmers, developers, interactive media production specialists, game “level” designers, musicians, content providers, project producers, digital artists, 3D modeling technique artists, tools and technologies programmers, project managers and test and quality assurance specialists.

It now takes entire teams composed of managers, artists, musicians, interface design specialists, system architects, developers, test engineers and quality assurance specialists to produce a game which increasingly looks like a movie production in scope, scale, and enterprise. Many of the most popular games are developed with multi-million dollar budgets and 12-18 month development life cycles or longer. Recalls are just not practical, especially for the console side of the industry with its solid state cartridges and fixed CD-ROM technologies.

And for those of you who remember *PacMan* or even played the arcade game, the world’s first perfect score was achieved on *PacMan* in New Hampshire in the United States earlier in 1999. Taking nearly six hours to accomplish the feat—on one quarter—Billy Mitchell, 33, a Fort Lauderdale, Florida U.S. hot sauce manufacturer visiting the famous Funspot Family Fun Center in Weirs Beach, New Hampshire in the U.S. scored 3,333,360 points—the maximum possible points allowed by the game. The results will go into the 1999 edition of the *Twin Galaxies Official Video Game & Pinball Book of World Records*—which is the official record book for the world of video

game and pinball playing. To achieve a perfect game on *PacMan*, the player has to eat every dot, every energizer, every blue man and every fruit up to and including board 256, where the game ends with a split screen.

So what can computer and video games development reveal about software quality? Could we actually learn something from *PacMan*? By examining what quality development and testing means in the context of the computer and video games entertainment industry we may learn something outside of what we already know or think we know about quality and testing on our side of the industry in information technology and software development.

Some of the findings may surprise us when we more closely examine how this fascinating segment of the industry actually performs quality and conducts testing under rather interesting market conditions and shifting consumer tastes and preferences and ever more powerful game consoles, CPUs, computers, and 3-D video cards.

Even in the mass-market world of computer and video game software, a lot of successful game development companies do not write detailed requirements and specifications, many do not freeze their designs early in development, and many do not even follow a traditional life-cycle. So how do they build quality in?

It turns out that the best developers and publishers in the computer and video game industry are using engineering discipline on a wide scale with dramatic results. They are creating and applying processes in the development of game software with the explicit goal of improving the quality, the maintainability, stability, extensibility, understandability and visibility of the software. The ability to consistently deliver on these particular engineering elements and still produce a great game is the key difference between the average and the superior game development companies.

The primary mission of most developers is to ship a “cool” game and make money and have the creative and emotional satisfaction in knowing that someone if not thousands or even tens of thousands of people are playing your game. The desirable shipping date for most games is dictated by the realities of the retail market and promotions.

In general most game software is not written to be maintainable. The primary goal is getting the code finished “enough” in order to ship it as a commercial product. Even many game developers still believe that documentation and coding standards come at the expense of getting the game out the door sooner. This mindset has worked for some time when teams were relatively small and time to market was one of the more important issues. But times have changed and it is the high quality computer and video game companies who have seen the light and run with it.

Why? The software development methods as employed by some of the game publishers and developers were simply not working very well. Product life cycles have increased dramatically and maintenance costs have become prohibitive. Most games are not financially successful, most even miss at least their announced shipping dates and many have significant bugs and a lot require multiple patches and upgrades.

So the high end game publishers are employing risk management and engineering discipline in an environment that is constantly changing due to consumer tastes, changing operating systems, competitive products, and marketing management fiat.

The paper will examine some of the key foundational elements and characteristics that make a good game succeed. Much of the background and research for this paper is based on industry analysis, numerous informal discussions with game developers and industry contacts, and my own software

and information technology process and product development experience and enjoyment derived from actual “playing” computer and video games.

Game Play

So what is this “thing” called “game play” and is there an adequate way of defining it let alone describing it? One of the keys for producing a high quality video console or computer game is understanding the intent and outcome of the final results first and then designing and building the game to provide those results. It all comes down to what the game player can see and experience as part of the “game play” experience. If they cannot tell it is happening, then it might as well not exist.

Many people in the computer and video game industry have different definitions for “game play.” Some will use the word playability. Others will list some of the more common attributes generally assigned to recognized or successful games like “addictive,” “easy to learn” but harder to master, “customizable,” “fun,” “engaging,” “enthralling,” has multiple “replay” value and more.

But a good game probably has a great sense of timing, and the creation of an atmosphere and presence for the gamer where imagination can become bigger than the game itself. Others have described “game play” as an essentially unknown quantity that is apparently realized when a game sells well and consumers rave about it.

Ultimately the final responsibility for the quality of the game and the game play itself lies in the hands of the producer with the designer having a major part in the computer and video game equation. It is the designer that is usually the most concerned about “playability” even before the first line of code is ever written. But the designer and the producer in these days of multi-million dollar game development budgets are usually not the same person. Games can be tweaked, enhanced, and changed during the testing phase but if the basic design of the game is flawed it is probably too late or there will be significant delays in the release of the game.

Game Design Overview

Again, a good design is fundamental for the development of a high quality game. Some game developers have even been known to build the player attributes into a table structure and then give the testers access to the same table when checking out various characters and character attributes. This type of approach allows for the testers to “feed” their improved attribute combinations and play experience to the developers who in turn can concentrate on bug tracking and fixing and performance tweaks and enhancements.

Game algorithms are not just neural networks and learning systems and complex mathematical structures, but are primarily centered on creating an environment and the appearance of “thought” from software units. Game play at the code level is much more about replicating behavioral models rather than scientific “reality” and this focus may reveal some lessons for software engineering and quality at large.

Game State Machines and Impact on Design

Interestingly enough, the best video and computer game development companies use state machines for building much of their “product.” Finite state machines are natural for any type of computer program and understanding how to use them effectively to create a computer game is usually dependent on the level of understanding of the system you are trying to represent, which is as detailed or as simple as you make it. There are many purposes for using finite state machines in games but one of the more intricate ones involves modeling unit-level behavior since trying to simulate human beings is arguably one of the toughest simulations around.

While some other types of systems are designed to more accurately model the way humans think and learn, sometimes you cannot exceed the simplicity of having a choice, weighing the factors and deciding, as a human, which one you would make given that choice. Therefore, game design becomes critical based on the intended outcome.

When it comes to high quality computer or video game design the best “solution” is sometimes the simplest and not usually the most scientifically accurate. In many cases, clever routines and interesting algorithms such as neural networks and genetic algorithms are not necessarily the best solutions if they will not give better results. Choices are often based on what is needed to obtain the end result and the gaming “experience,” instead of just mimicking the latest trends. The more of these details that are anticipated through planning and testing for the game, the more immersive the environment and experience for the game player during game play.

So how does one begin to produce a game? Even in the computer and video games industry software development is not easy, but there is a lot more work done in the design phase of a game than in the coding phase.

Creating a Computer or Video Game

It is possible to make a game quickly. It is also possible to make a game cheaply. It is even possible to make a game that is great. But even in the computer and video game industry you can only pick two of them. To make a great game quickly will cost a lot of money. A large and experienced development team will cost money. To make a game cheaply is possible but may take some time to find bargains on people and technology. A cheap game made quickly will probably flop in the volatile consumer marketplace.

Computer and video game development is really a marriage of creativity and technology and the combination of these two elements add a high degree of uncertainty when planning the budget and schedule for the project. Risks include the following:

- Creative game programmers and designers will suffer from “writer’s block” and artistic and emotional temperament changes so productivity is affected from time to time
- Skilled developers and designers are in great demand and there are many opportunities in the computer and video game industry. Sometimes they move on.
- Technology changes very rapidly in this industry and it is frequently necessary to make unanticipated course corrections during the life of the development effort and particularly in the middle of the development effort

- Developers and programmers are often asked to invent new algorithms or techniques during the course of the project and it can be difficult to accurately predict, let alone always plan for new innovations and technology updates

Even in the computer and video gaming industry there are still many business people who cannot understand why experienced developers cannot make games on time and on schedule. And many experienced developers do! Developing games is no longer a hobby, it is big business and getting bigger. Many game publishers and developers are adopting many high quality software production techniques and practices combined with the necessary process discipline for high quality development plus some very ingenious state based genetic algorithms and behavioral models.

Legacy Issues in Game Development

Due to market cycle time and game product shelf lifecycles many computer and video game publishers and developers will understandably rely on legacy code, legacy engines, and legacy functionality. However, due to the relentless march of technology many of the legacy programming assumptions can cause more problems during future development. CPU speeds, graphic processors, 3D hardware, 3D algorithms, and 3D engines are evolving so rapidly that there is a real danger of carrying the assumptions and techniques from the current product into the next. Programmers and game publishers will naturally feel more comfortable with techniques and tricks that are already mastered but developing for leading edge technology is a delicate balance of functionality, performance, design and workflow, so doing better with less in a better way is the new rule for optimization and performance.

High quality game developers continually fight code entropy – the process by which code becomes static and unwieldy and full of patches and modifications. If a new fundamental assumption comes along then the associated code must be thrown away and rewritten. Incremental patching and updating may seem easier at first but in the end the associated game software becomes much more difficult and usually results in the production of less robust, more error prone, and inferior code in the long run.

High quality computer and video game development methodology increasingly recognizes that it may seem safer to modify existing code that works but the nastiest and most pernicious of bugs almost always show up in patched up code and NOT in code designed from the ground up. It is hard work to do the upfront design but in the end it is worth it because the code is simple, elegant, and just plain right the first time – not the 10th or 11th time!

High Quality Game Development – It Really Is All in the Design!

The first critical stage in game development is working out the design and writing it down. Starting to code right away might seem like the quickest solution but sooner or later you are going to hit design and programming snags. Why? Because it very important to completely think through the game ideas. For extreme quality game development, designing and coding are two separate tasks that deserve equal attention.

Like a cook preparing ingredients, a good designer lines up the details of the graphics, sounds, and music that are typically needed during the design phase. Creating a game requires a lot of non-

code items. And the development team needs to have a good idea of what those will be and how they will acquire or create them.

This is also where quality really begins in game development. Many producers in the game industry still think that quality assurance and testing is something that is done before they go to beta release. In this mindset testing begins somewhere between the alpha and the beta release. The basic misconception is that testing and QA are only done after the product is testable. In truth, testing and QA as practiced by the successful game development companies are an indispensable and even crucial part of the complete game development cycle from design to release and it begins with design.

The design is the blueprint for the game product. And guess what? A good QA engineer can take a well thought out and considered design and create test scripts, automated tests, and all kinds of ad hoc and user based scenarios necessary to shake down the game. A design for a game can be as simple as a single end to end event driven thread or involve the complexity of a “state” map with all of its possible outcomes.

The design usually takes operational form as a running set of sketches, short essays and notes in the form of the storyline and the interface. Game designers are constantly adding to it until a nearly complete picture of what the game will be begins to emerge, always being careful not to do too much.

Beginning designers and novice game development teams often complain that a detailed design is a waste of time and effort. Experienced teams do not. An experienced team recognizes that it is always more efficient to think through what’s necessary before you actually program the game rather than thinking about it afterward, when the thinking is influenced by what was programmed. A good design document will answer the question posed eight months later by someone asking about what is needed in a particular section of the game.

The design document also gives the designer the “authority” to promote team interaction while retaining creative control. The design document requires participation from the programmers, the artists, the content creators and other team members and any additions or modifications are incorporated into the design document as part of the peer review process. Anyone that wants to make a change needs to coordinate with the designer who is in the position to incorporate the approved changes in a logical and consistent manner while maintaining the integrity of the player interface, game concept and that all important game play.

The design phase includes all of the key game project team members but the designer(s) retain “concept leadership” in order to prevent the loss of focus and maintain the integrity of a consistent design. A consistent design which “holds” the game framework together is usually superior to the far greater risk of including a random and chaotic set of personal preferences.

Most design documents in high quality computer and game development have a “middle” period. This period usually revolves around finishing the design document and maintaining the design document and the design concept. But in the end, the most important thing to consider when developing games is to finish them. A finished game with less flash and dazzle is better than no game at all.

A high quality game will usually succeed if it is unique either by storyline, game dynamics, or even graphics. In order to have any chance of success a game usually must provide a discernible

difference from everything else out there. Remaking a clone of Asteroids is not a good way to go unless you are into “retro” or classic gaming!

Even with a solid design not every game is a best seller given the level of sophistication, polish, and uniqueness that such a product requires combined with the buying vagaries of the consuming public. Even if a game is not up to these standards there are many outlets beyond the retail realm for showcasing and receiving credit and recognition. Uniqueness is one way. Being unique is a general code word for simply offering the consumer of your game a specific reason to choose that game when they compare it to others. Design is fundamental in this equation.

Fundamentals of Game Design

So what happens once coding does begin for a game? All of the steps in the design stage apply to coding plus a few more. A typical mistake made by many game programmers is not commenting the code. Game programming requires a lot of clever work arounds, much more so than other forms of programming, and requires serious optimization many times late in the development cycle. This means commenting is even more important for games. Creating code that can be applied to other games later demands well-commented and very open ended and modular code.

Everyone has ideas that need development. Games are perhaps one of the most creative mediums ever. They require music, sound, art, storytelling, writing, programming, and more. This requires a lot of brainstorming. Every creative person needs to come up with ideas, and with games even more so, as this medium has perhaps the most fickle, demanding, and even unforgiving customers in the marketplace. The key is to be very open and write down whatever comes to mind. Most game designers and developers try to focus on different things, such as adventures they have had, non-game things they like, movies they have seen, and books they have read. Many developers are comparison developers and are constantly in stores reading the backs of the game boxes, evaluating demos, and just playing other games.

In a creative medium like this, borrowing someone else’s ideas is how a lot of them create. When a new game is written, it attempts to incorporate (rip off) all of the current ideas out there and then move beyond them, only to create new features which themselves are incorporated into other games. This is how games evolve. Many developers also spend a lot of time reading non-technical materials. A high quality programmer of games is in many respects a renaissance person. They must have an understanding of many different elements involving the arts, technology, and the general world around them. How can one write a game about geopolitics if they haven’t read about or understand the world around them?

So just what is game design? Game design concerns one thing above all else and that is interaction. What separates games from similar creative mediums like art, movies, music, and books, is that the player interacts with the medium. You don’t stare at, or just listen to a game. You control and interact with the game. Therefore game designers and producers have the challenge of creating a product which entices people to play, and at the same time, provide the storyline, the emotional feel, the realistic tone, and the other qualities all other creative mediums give us. It is clearly a tall order, but this is what makes creating games so much fun for most game developers.

Playing means making decisions. Therefore the design of most games demands the creation of situations and scenarios where the game player decides what to do and performs that action. This can be as simple as *PacMan* where the player has to decide whether to go UP, DOWN, LEFT, or

RIGHT using their joystick or keyboard or as complex as *Civilization II* juggling the myriad of variables in establishing a nation or country as the leader.

This is what makes games so appealing – a set of decisions which the player controls and, based on their skill and intelligence, by which they ultimately decide the outcome of their game. The game designer tries to provide an easy way for the player to make decisions during the course of the game, and then provide interesting outcomes that in turn lead to new situations and the whole process starts over.

Most high quality game designers and programmers try to keep the following things in mind in the creation and development of a game.

- Is there interaction?
- Does my design create a decision dilemma for the player or not?
- Are clear situations provided to the player?
- Is there enough information in the game (graphical/sound/text) to illustrate to the player what situation they are in?
- Are they provided with the proper information to make decisions?
- Is the interface by which the player commands the game clear and easy to use?
- Does it provide the proper information to them to help them input desired actions?
- Do the outcomes of the player's decisions end or continue the game?
- Does the skill and intelligence of the player produce the outcome? Random outcomes not based on the skills of the player's decisions are not games. Players must know they are controlling the outcome.
- Is it entertaining? If it isn't fun, they won't play it.

In short, the best game developers concentrate on providing interaction, creating player control of their outcomes based on their skills and intelligence and make it fun.

Game Coding

So how does a game developer turn ideas into finished games and finished games into products that can be sold in the consumer marketplace? The dominant language of game development is C and C++. Almost every game you see is written in these languages. C and C++ are also good languages to write in because it is relatively easy to port the C or C++ code from one platform to another. Even though “easier” hybrid languages exist, C and C++ are still the dominant game development languages. Many sophisticated but integrated development environments have served to make it even easier.

What about the use of Assembly language? Since it is the fastest language, some Assembly language is used in game development. Assembly is usually used to create subroutines to call from C or C++ for sections requiring intensive speed. Assembly language is the most difficult to understand. The general law of computing languages states: *The lower the level of the language, the faster it is, and the harder it is to program in it.* With its portability and easier learning curve, C

and C++ are much easier than Assembly. However, no one said programming in C or C++ was easy either, just easier than Assembly and some game developers even work with Visual Basic and even other languages and scripts. With the advent of the Internet and online gaming, much of the multimedia content and game applications are increasingly written in Java, JavaScript or Visual Basic and even variants of LISP!

So what are some of the programming basics and design fundamentals for high quality computer and video game development? Part of what is fundamentally necessary beyond graphics, core programming, and animation creation centers on artificial intelligence. Artificial intelligence centers on the creation of intelligent reactions by the game of the situation and the decisions and is most commonly used to create computer opponent assessment and decision logic trees. It is not an arcane art and game programmers have developed many established methods.

TESTING and QA

So how do the game developers and producers test and ensure the appropriate level of quality assurance? A buggy game released too early to market is very risky and the gamer will quickly spread the negative word around. Before the game developer shows the world their game they need to make sure there are no bugs or problems. Even moderate beta testing will let the developer know if there are any problems prior to releasing it in order to make sure it runs properly.

Testing is a huge and vital part of the computer and video game development process. In fact, most games now fully credit the entire development team and most game development efforts have dedicated quality assurance and testers. In conversations that I have had with video and computer game developers all of them without fail emphasized the critical and indispensable role that QA and test plays in getting it right before it goes out the door. So how do video and computer game producers and developers “play” test their code?

It is usually done in one of two ways. The first is through the use of actual consumers and observing them in a usability lab or room or by sending out “beta” copies of the game and soliciting feedback in the form of e-mail, a questionnaire or via a hot-line phone number. Some of the more innovative producers and developers have even been known to use Internet Relay Chat (IRC) to obtain feedback! There is also something called “focus” testing which is usually conducted by the marketing team and usually involves showing an early release or demonstration version of the game or game concept to a targeted group and getting feedback. But by far it is “in house” testing and quality assurance combined with temporary play testers that bear the brunt of play testing. This type of testing usually requires much more formal observation and recording of the results.

What are the different types of testing? Many of these are quite familiar to those on the information technology software development side of our industry, including unit testing, component testing, white box versus black box testing and even gray box testing, system testing, and integration testing. So what is different about how testing is conducted for computer and video games development?

First is compatibility testing. Most game publishers and developers save for the very largest cannot afford to have all of the hardware platforms around for testing and many contract out. It is a bit easier with the console systems with their single architectures. It is also usually the case that the entire development team has been working on a high-end machine. This does permit faster and

more rapid development, but then the game has to run on a minimum configuration even if a maximum is recommended.

Chipsets are also another issue in testing. Does the game rely heavily on 3D technology? What about sound drivers? Music formats? What about multiple drivers for certain types and kinds of cards? Bugs found early during this testing are less likely to end up as features later.

High quality game publishers also practice extensive functional and automated testing. This works best for those games just after alpha release where the game is predictable and the functionality is essentially locked. Regression testing each internal release provides for an integrity check of the design and any code changes or additions that were made since the last baseline. Stress and load testing is also used for uncovering various types of resource bugs and memory leaks.

There is also ad hoc testing. This usually takes the form of version play testing. Testers can try different things that the developer or designer had not considered. Many undocumented features have been discovered using this type of testing. A form of ad hoc testing is *dirty* testing. The intent of dirty testing is to break the software. This usually involves revealing holes in the interface, things the designer or the programmer had not considered and attempts to simply crash the game and uncover “show stopper” logic faults.

Many games today are built with scripting engines. But as is usually the case, the script changes. Depending on the particular scripting engine, adding some extra functionality can break the script. The way the functionality is called can break the script during modifications later. A good QA and test team will test the engine by deriving test cases directly from the scripting engine.

Play Testing – The Heart and Core of Computer and Video Game Testing

The biggest difference between computer and video game testing and traditional software testing is probably in “play testing.” The timing for conducting play testing is very important. The game must be stable enough (sound familiar?) so the QA or play tester does not have to spend too much time noting all of the instruction level or operational bugs, yet immature enough to allow for those design and logic changes that can still be made in time.

Initial play testing usually involves setting the game to the shortest interval if that is possible. If it is a graphically oriented adventure game then several different scenarios or environments are executed. In addition, “coverage” play testing is critical and this includes full end to end event triggered functionality. If there are bonus or non-linear environments, shortcuts are usually provided so that these environments are also fully tested. Not testing all the event-triggered paths can result in games that do not behave correctly. For example, a game called “Impossible Mission” was released back in the 1980s for the Atari 7800 game machine and it was literally impossible to finish as the logic for the final mission was not provided as part of the live executable.

But what if the testers at this point in the development cycle find that the game is not up to par? This is the time that QA usually calls a strategy meeting with the production, design, and test groups and they all review the “usability” results and analysis. Some of the questions that are examined would include the following:

- Are the testers complaining about the same things that earlier testers complained about?

- Are there still bugs and problems in the game? If so, these bugs will come back to haunt the team in the form of the very public product reviews in the trade magazines and on-line chat and newsgroups!
- How long does it take before the play testers are bored with the game? A good testing schedule in the video and computer game industry involves a dinner break every 4 hours and 15 minute to half an hour breaks every 2 hours or so. After a week of play testing a formal group session with all the critical personnel is typically held to discuss the outcome. Of course there are many informal sessions in between.

So what does play testing really provide as value and as a benefit? Play testing should provide the producer with as much information and data as possible for making all of the necessary game tweaks and adjustments. This type of testing is designed to provide much more information than just lockup and crash problems. Bug reports typically are categorized by severity but also include an area for subjective feedback and opinion or comments. Why? It is the test department that is playing the games as their livelihood. They are paid to sit and play the games all day! When a significant number of play testers are literally begging to take a copy home or are staying and playing for hours on end there is probably a good game to market.

The Critical Role of QA

So what is the real role “played” by QA in the development of a computer or video game? The primary objective of QA is in really making sure that the right “mix” of testing talent is available. They are also the binary no/go gates for that all important release quality. They also ensure that the vital information and feedback loops are running at or near full bandwidth on the project and are constantly pulsing the project in order to make sure it meets its goals.

It turns out that turnover for testers is fairly high in the industry so QA must be able to identify and hire and retain skilled testers. What are some of the necessary attributes for a qualified tester? They include excellent oral and written communication skills. These people side skills are fundamental and the “glue” that makes all the difference. If you hire someone who cannot write understandable bug reports then communication is disabled. A qualified tester should also possess a variety of video and computer gaming experience and domain knowledge. Game genres include sports games as well as simulations and puzzle games and educational games. However, testers with less experience in these areas are still valuable as they can concentrate on the interface and playability issues that many game experts might just take for granted or even overlook. The producer and QA’s biggest task is taking all of the bug reports and “play” reports and figuring out what will make the most impact on the game with the least disruption upon the design and the game’s release schedule.

What kind of environment is ideal for game play testers and QA game engineers for the most successful of the game development companies? Testers and QA personnel are taught to stick to their opinions and feedback. This is true even if the producers due to marketing pressure or schedule constraints attempt to prevent them from logging bug reports or “negative” feedback. Some producers have been known to go to great lengths to get their game through testing and it is testing and QA’s job to report ALL of the issues that are important. But testers and QA cannot become hostile either with the programmers and developers, even when sticking to their guns. If a

tester is perceived as something other than thoughtful or considerate they may get very little if any cooperation from the developers.

Testers should give both the negative and the positive feedback without having to worry about how the producers or the developers will react or feel about it. The whole team is ultimately responsible for the quality of the game but it is QA and the testers who are accountable for that quality during development. Testing is designed to find errors in game play, the functional logic sequences, and in the interfaces and this information and feedback is vital in the managing of the game development and game production life cycle.

So what does a typical test group look like in the computer and video game industry? Generally, testing teams consist of one lead, an assistant lead and 3-8 full-time testers depending on the type and complexity of the game. There is usually a testing lab and they are usually located in an area that encourages and facilitates communication and information exchange among and between themselves. Note that many game testers are not actually trained in formal software testing methodologies and most of their test training is acquired on the job. Again, does this sound familiar?!

The assignment and location of testers and QA personnel with the project developers is generally not encouraged as it often hinders objective feedback. But the testers do need to communicate with the developers and to understand the fundamental architecture and game play requirements in order to discover and identify the bugs. The ideal testing environment for a computer or video game in this industry is a combination of intense game play, communication, and discussion dedicated to eliciting and soliciting game play oriented comments and feedback.

The Importance of Teams

What was once the province of one or two people is now the province of 25, 30 or even more for the number of full-time people working on a game project. Development times are now 1 –1.5 years for arcade style games, while much more complex games can take from 18 months to 2.5 years or even more. Game development is much more like film and TV production and the development teams are composed of many different specialists. The mission of the team is to create a great product. The team's management has the job of creating and maintaining the team. Based on informal interviews and research conducted by the author some key high quality management process practices characteristic of teams in the computer and video game industry have emerged as follows:

- Bottom up approach/top down design – Team members are asked what they can do for a schedule instead of telling them what to do.
- A planning phase takes place before one line of code is ever written.
- The design is completely shared with the team. The designer(s) listens and questions as the functional specifications are created. All team members are represented in these meetings.
- All team members know what they are supposed to be doing from day one, every day.
- There is a built-in and managed QA feedback loop, not just for after delivery but throughout the development lifecycle so problems can be caught while there is still time to fix them.
- Writing down and recording all the things the project team agrees on

The Importance of Engineering Process and Practice Discipline

High quality game development and delivery is fundamentally centered on engineering discipline both at the management level and the technical level and still providing for the necessary freedom to create. Some of the keys are programming standards, peer reviews, and process working groups.

The importance of programming standards cannot be easily overlooked. They fundamentally make sure that poor coding practices are not showing up in new games. The payoff is in maintenance and improvements.

Peer reviews permit the removal of defects from the software product early and more efficiently. A side effect of peer reviews is creating a better understanding of the game product elements and the defects that might actually be prevented. Peer reviews are fundamentally a mechanism for development level process improvement. The primary value of peer reviews permits the discovery of problems and potential problems before they are found in the testing stage. A less visible but very important alternative benefit is that all the participants in a peer review process learn to avoid both common and subtle errors in their own work.

Peer reviews are never part of the personnel review process. The high end game publishers have discovered that peer reviews are finding important problems that have traditionally defied other debugging and testing methods. As people gain experience with the peer review process many find that they are uncovering problems earlier and earlier in the development process and uncovering bugs that had traditionally taken many weeks during beta testing.

Another key process activity that the high end game publishers have adopted is process working groups focused on game concept, requirements definition, design, interface and implementation, and maintenance. All of these processes have feedback mechanisms designed for continual improvement.

Software testing plans are initially created in the design phase of the process and then completed in the implementation phase. High quality software test plans describe both the methodology and the elements that make up the actual tests for the games based on the requirements and the game play.

What are game publishers finding as real and lasting benefits from this process? One of the biggest is morale. One of the more important benefits is having all the project baseline requirements written down and agreed to by all key team personnel. When a requirement change occurs, the QA team can document the change and inform all parties concerning the impact of the change. It also permits informed cost and schedule tradeoffs and reduces the *chaoticness* of feature creep. Even thinking about the requirements and the associated specifications heads off problems down the road.

When the development team has a full conceptual design that is documented, prototyping technology available, and a full tracking of all tasks plus the full creative input from the graphics, content, scripting engine, sound, music, and development engineers and QA, then the management of all the combined resources is much more complete and effective, with much fewer variables, and less subject to surprises.

The key for quality is ultimately reducing the time it takes to find bugs. By completely planning the product prior to implementation, through the use of peer reviews, and implementing release go/no go gates then the computer and video games industry will continue to raise the bar of quality.

One of the real benefits of this process has allowed many game developers to experience a normal day instead of grueling 16 hours days or longer ending up in a project death march. Team members now have the opportunity to contribute in many other ways because the environment and the time support it.

And finally the cowboy coder mentality is no longer necessary or even desirable. The entire team works together for a common purpose without the need for last minute heroics. It takes much of the chaos, vagaries, variabilities, and uncertainties out of the picture and allows the team to really focus on producing a game that satisfies the game player, generates revenue and creates great game play.

Conclusion

The world of computer games is as competitive a business as there is these days. Working 60-80 hour weeks, for as long as two years or more is no longer unusual. That is the norm in any extremely competitive business. Since getting an advance is tough to do, a typical developer puts in a lot of hours. Just a few years ago, people could make it in this business without making that kind of sacrifice, but that is not possible any longer in most cases.

Developing a high quality game that receives critical acclaim and makes money can be brutal given the seriousness of the business and the time it takes and the competition. Game development, unlike a lot of other programming, is at the forefront of the technology curve. That is why you see games pushing the hardware limits of computer and video console games more than the typical word processor. So much of what we see done today was figured out by someone only yesterday.

The bottom line in computer and video game quality is centered on achieving the best results, not necessarily having the most fashionable routines. The game developer mantra is, "If it looks right, it is right." Computer and video games are never perfect replicas of reality. Given the extreme nature of developing computer and video console games in this industry the trick is drawing a line for "good enough" and then making that "good enough" reality. The best in breed in the computer and video games industry depend very heavily on QA and test to make sure that the game really is "good enough."

PTR Prentice Hall, 1997.

Maguire, Steve. *Debugging the Development Process: Practical Strategies for Staying Focused, Hitting Ship Dates, and Building Solid Teams*. Microsoft Press, 1994.

McConnell, Steve. *Software Project Survival Guide*. Microsoft Press. 1997.

McConnell, Steve. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, 1993.

McConnell, Steve. *Rapid Development: Taming Wild Software Schedules*. Microsoft Press, 1996.

Bennatan. E.M. *On Time, Within Budget: Software Project Management Practices and Techniques*. John Wiley & Sons, 1995.

Magazines

Game Developer, Miller Freeman, Inc. <http://www.gdmag.com>

Newsweek, "Who's Got Game?" 6 September 1999. <http://www.newsweek.com>

Next Generation, Imagine Media, Inc. <http://www.next-generation.com>

PC Gamer, Imagine Media, Inc. <http://www.pcgamer.com>

Wired, <http://www.wired.com/wired/current.html>

Industry

Classic Gaming Expo '99. Informal/Non-Attribution Interviews and Discussions with various computer and video game developers and publishers. Las Vegas, Nevada, 14-15 August 1999.

Extreme Quality - What Can We Learn from the Computer And Video Games Industry?

EXTREME QUALITY: What Can We Learn from the Computer and Video Games Industry?

Thomas Drake
Quality Architect/Software Archaeologist
Enterprise Management and Information Technology Consulting
Coastal Research & Technology, Inc.
Quality Week Europe 1999
tadrake@earthlink.net

© Copyright 1999 by Thomas Drake. All Rights Reserved



Quality - Software Engineering's Middle Name

Thomas Drake 1



Extreme Quality - What Can We Learn from the Computer And Video Games Industry?



The Computer and Video Games Industry

- ☑ Big business and getting bigger!
- ☑ Projected revenues in the U.S. of \$8 billion for 1999
- ☑ Sony, Nintendo, Sega, Macs, PCs, handhelds
- ☑ Incredible technology - creating the envelope
- ☑ Remember PacMan?!
- ☑ What can we learn from this industry for quality?




Quality - Software Engineering's Middle Name

Thomas Drake 2





Slide 3

Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




An Example of Moore's Law in Action!




Quality - Software Engineering's Middle Name

Thomas Drake 3






Slide 4

Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Game Play

- ☑ The heart of the game
- ☑ All about intent and outcome
- ☑ Attributes for a good game
- ☑ Importance of design



Quality - Software Engineering's Middle Name

Thomas Drake 4



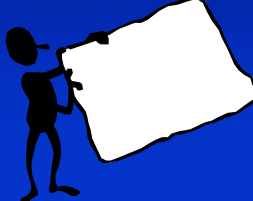
Slide 5

Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Game Design Overview

- ☑ Design is fundamental!
- ☑ Game algorithms
- ☑ Behavioral modeling
- ☑ Attributes - The key for testing



Quality - Software Engineering's Middle Name

Thomas Drake 5



Slide 6

Extreme Quality - What Can We Learn from the Computer And Video Games Industry?



Game State Machines and Impact on Design

- ☑ State is everything
- ☑ Contribute to the gaming experience
- ☑ The greater the details, the greater the experience
- ☑ Best solutions are the simplest - not necessarily the most scientifically accurate




Quality - Software Engineering's Middle Name

Thomas Drake 6






Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Creating a Computer or Video Game

- ☑ Pick two!
 - Quickly, cheaply, great (fast, cheap, quality)
- ☑ A marriage of creativity and technology (risk)
 - Creative game programmers and designers will suffer from "writer's block" and artistic and emotional temperament changes so productivity is affected from time to time
 - Skilled developers and designers are in great demand and there are many opportunities in the computer and video game industry. Sometimes they move on.
 - Technology changes very rapidly in this industry and it is frequently necessary to make unanticipated course corrections during the life of the development effort and particularly in the middle of the development effort
 - Developers and programmers are often asked to invent new algorithms or techniques during the course of the project and it can be difficult to accurately predict, let alone always plan for new innovations and technology updates






Quality - Software Engineering's Middle Name

Thomas Drake 7



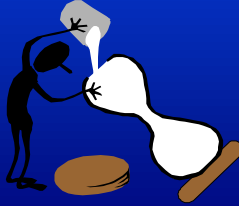



Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Legacy Issues in Game Development

- ☑ Issue of market cycle time and shelf life
- ☑ Problem of code entropy
- ☑ Impact of incremental patching and upgrading
- ☑ Doing it right the first time - design is everything
- ☑ Simple elegant solutions - stand the test of time and the marketplace

Quality - Software Engineering's Middle Name

Thomas Drake 8



Slide 9

Extreme Quality - What Can We Learn from the Computer And Video Games Industry?



High Quality Game Development - It Really is All in the Design!

- ☑ Writing it down
- ☑ **Ingredients**
- ☑ Becomes the master blueprint
- ☑ **QA role - creation of test scripts, automated tests, ad hoc and user based "play" scenarios**
- ☑ Operational form of design - sketches, short essays/stories, notes, interface details
- ☑ **Promotes team interaction/creative control**
- ☑ **Concept leadership and focus**




Quality - Software Engineering's Middle Name

Thomas Drake 9 


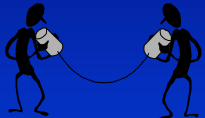
Slide 10

Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Fundamentals of Game Design (1)


- ☑ Comments and feedback
- ☑ **Interaction, interaction, interaction**
- ☑ Decision situations and scenarios (PacMan/Civilization)
- ☑ **Some desirable design considerations**
 - Is there interaction?
 - Does my design create a decision dilemma for the player or not?
 - Are clear situations provided to the player?
 - Is there enough information in the game (graphical/sound/text) to illustrate to the player what situation they are in?
 - Are they provided with the proper information to make decisions?



Quality - Software Engineering's Middle Name

Thomas Drake 10 


Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Fundamentals of Game Design (2)

- Is the interface by which the player commands the game clear and easy to use?
- Does it provide the proper information to them to help them input desired actions?
- Do the outcomes of the player's decisions end or continue the game?
- Does the skill and intelligence of the player produce the outcome? Random outcomes not based on the skills of the player's decisions are not games. NOTE: Players must know they are controlling the outcome.
- Is it entertaining? If it isn't fun or engaging, they won't play it!


☑ In short, the best game developers concentrate on providing interaction, creating player control of their outcomes based on their skills and intelligence and make it fun.




Quality - Software Engineering's Middle Name

Thomas Drake 11 


Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Game Coding



- ☑ C and C++
- ☑ Integrated development kits
- ☑ Want speed with spectacular? - use Assembly!
- ☑ The general law of computing languages states: *The lower the level of the language, the faster it is, and the harder it is to program in it*
- ☑ Online gaming - Java, Javascript, Visual Basic, LISP
- ☑ AI - Decision Logic Trees/Assessment Tables




Quality - Software Engineering's Middle Name



Thomas Drake 12 

Slide 13


Extreme Quality - What Can We Learn from the Computer And Video Games Industry?



Testing and QA




- ✓ Get the bugs out!
- ✓ Indispensable for high quality development/delivery
- ✓ Usability labs/focus testing
- ✓ In house testing - unit, component, white and black box with gray, integration, system, compatibility, chipsets, functional, automated, regression, ad hoc, dirty testing (break the software), stress and load testing (resource constraints/bugs and memory leaks)
- ✓ Deriving test cases from the scripting engine




Quality - Software Engineering's Middle Name

Thomas Drake 13





Slide 14

Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Play Testing - The Heart and Core of Video Game Testing

- ✓ THE most important testing -
 - > The game must be stable enough (sound familiar?) so the QA or play tester does not have to spend too much time noting all of the instruction level or operational bugs, yet immature enough to allow for those design and logic changes that can still be made in time.
- Are the testers complaining about the same things that earlier testers complained about?
- Are there still bugs and problems in the game? If so, these bugs will come back to haunt the team in the form of the very public product reviews in the trade magazines and on-line chat and newsgroups!
- How long does it take before the play testers are bored with the game?




Quality - Software Engineering's Middle Name


Thomas Drake 14






Extreme Quality - What Can We Learn from the Computer And Video Games Industry?



The Critical Role of QA




- ✓ The primary objective of QA is in really making sure that the right “mix” of testing talent is available and that the right amount of quality is present
- ✓ They are also the binary no/go gates for that all important release quality.
- ✓ They also ensure that the vital information and feedback loops are running at or near full bandwidth on the project and are constantly pulsing the project in order to make sure it meets its goals.




Quality - Software Engineering's Middle Name

Thomas Drake 15





Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




The Importance of Teams

- ✓ High Quality Practices
 - Bottom up approach/top down design – Team members are asked what they can do for a schedule instead of telling them what to do.
 - A planning phase takes place before one line of code is ever written.
 - The design is completely shared with the team. The designer(s) listens and questions as the functional specifications are created. All team members are represented in these meetings.
 - All team members know what they are supposed to be doing from day one, every day.
 - There is a built-in and managed QA feedback loop, not just for after delivery but throughout the development lifecycle so problems can be caught while there is still time to fix them.
 - Writing down and recording all the things the project team agrees on




Quality - Software Engineering's Middle Name

Thomas Drake 16





Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




The Importance of Engineering Process and Practice Discipline (1)

- ☑ Programming Standards/Coding Standards
- ☑ Peer Reviews
 - Permit the removal of defects from the software product early and more efficiently.
 - Better understanding of the game product elements and the defects that might actually be prevented
 - The discovery of problems and potential problems before they are found in the testing stage
 - All the participants in a peer review process learn to avoid both common and subtle errors in their own work
 - The high end game publishers have discovered that peer reviews are finding important problems that have traditionally defied other debugging and testing methods




Quality - Software Engineering's Middle Name

Thomas Drake 17





Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




The Importance of Engineering Process and Practice Discipline (2)

- ☑ Process working groups -
 - Game concept
 - Requirements definition
 - Design
 - Interface and implementation
 - Maintenance/support
- ☑ All of these processes have feedback mechanisms designed for continual improvement.
- ☑ Software testing plans are initially created in the design phase of the process, completed in the implementation phase
 - High quality software test plans describe both the methodology and the elements that make up the actual tests for the games based on the requirements and the game play.




Quality - Software Engineering's Middle Name

Thomas Drake 18





Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Conclusion/Final Thoughts Have Fun Playing! (1)

- ☑ Developing a high quality game that receives critical acclaim and makes money can be brutal
- ☑ Game development, unlike a lot of other programming, is at the forefront of the technology curve
- ☑ So much of what we see done today was figured out by someone only yesterday
- ☑ The bottom line in computer and video game quality is centered on achieving the best results through creative game play




Quality - Software Engineering's Middle Name

Thomas Drake 19





Extreme Quality - What Can We Learn from the Computer And Video Games Industry?




Conclusion/Final Thoughts Have Fun Playing! (2)

- ☑ The game developer mantra is, "If it looks right, it is right."
- ☑ Computer and video games are never perfect replicas of reality
- ☑ Given the extreme nature of developing computer and video console games in this industry the trick is drawing a line for "good enough" and then making that "good enough" reality
- ☑ The best in breed in the computer and video games industry depend very heavily on QA and test to make sure that the game really is "good enough."



Quality - Software Engineering's Middle Name

Thomas Drake 20



Improving Quality in the Software Development Process on the Basis of UML

Arthur Görge, Minsheng Liu
Hüingsberg AG
Lilienthalsstraße 2
D-85399 Hallbergmoos, Germany

Abstract

This paper represents the result of the execution of the project funded by the European Systems and Software Initiative (OFTPIVE.PIE). In the project the object-oriented method with UML and a new paradigm approach have been introduced to the application environment of the software development and to improving the process of the software development. With the aid of the UML the weaknesses in the process of our software development have been recognised and the process has carefully been redefined in detail and improved. The quality of each step in the software development process has been controlled. It is shown that the introduction of the UML, CASE tool and a new paradigm is beneficial to the software development process.

1. Introduction:

Hüingsberg AG is an international company in the field of information and communication technology based on the Organisation for Data Exchange for Tele Transmission in Europe (ODETTE). In the past the company has developed, manufactured and marketed a wide range of hardware and software products in the ISDN area for the European automotive industry. We emphasised the importance of servicing for the customers and meeting customer's requirement, however we did not pay much attention to develop the methodology of our software product; we concentrated on the constructive improvement of software products and on the measurement of the intermediate products and end products, but we did not take much notice of the quality of the software development process, by which the quality of a software product is affected. As the complexity of products is increasing and customer's requirements are not constant, we realised that a permanent improvement method of a software development is urgently required and the quality of improving the software development process is important for the end products, if we want to provide our customers with better service and high quality products.

The goal of this project is to improve the software development process and to reduce the errors of analysis, design and implementation in the process of software development by using object-oriented methods in our application environment. So we are able to provide quickly new products responding to customer requirements with high quality and with the best cost-to-benefit ratio. In this paper the project of OFTPIVE.PIE is firstly described in brief, then we report the results about the improvement of the software development process by introducing the UML, CASE tool and the new paradigm approach.

2. Description of the Experiment

This section describes briefly the baseline project and presents the definition and the plan of the OFTPIVE.PIE project.

2.1. The baseline project:

In the European automotive industry file transfer based on ODETTE File Transfer protocol uses simple point to point links on ISDN, today. There is a number of file exchange systems between different manufactures and suppliers. As the CAD transfer volume is increasing, these systems can no more match the customer's needs. The application of electronic communication for file transfer is necessary and important for the automotive industry.

On the basis of the international networks and Corporate Networks the baseline project [3] has been developed for the file transfer in the automotive industry. The ODETTE file Transfer Protocol takes an interface function for a standardised and efficient communication over the network as shown in Fig. 1.

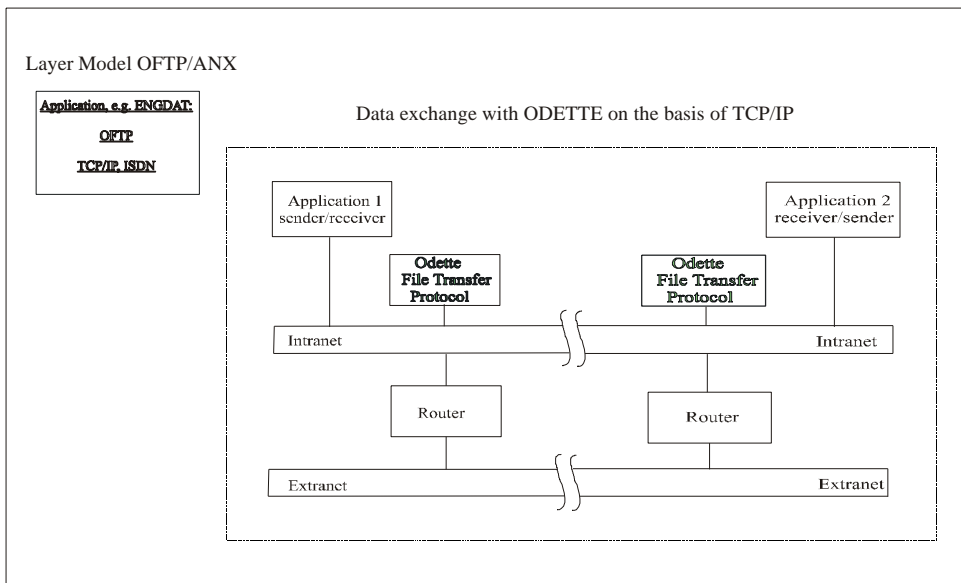


Fig.1: File transfer through TCP/IP in the baseline project.

The software product of the baseline project is based on logical connections between product providers and manufactures, so the software product provides our customers with much efficient service. In the experiment two server modules were designed in the conventional way using Microsoft C++ and Micro FoxPro as programming language. One transmission protocol is subject to the test-wise re-implementation by making use of object-oriented modelling and programming.

2.2. Definition of the OFTPIVE.PIE project:

The OFTPIVE.PIE project is proposed to evaluate a new software development method, namely, object-oriented method by using UML (Unified Modelling Language) and a CASE Tool for the software development process in automotive industry. The programming language JAVA is employed for the support environment.

In the PIE project we implement four modules of the OFTPIVE system in the baseline project. The experiment consists of the following components: Sender and receiver process, Logging activities, Interprocess communication and Data security considerations as shown in Fig.1.

2.3. Execution of the OFTPIVE.PIE project

In order to achieve the goals we have adopted the following activities:

- Organising an experiment team,
- training program for team members;

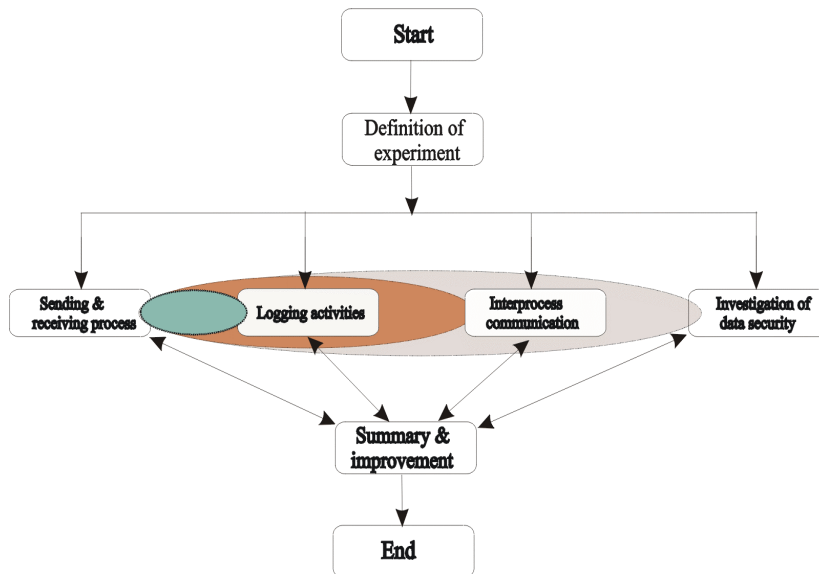


Fig. 2: Composition of the PIE experiment and iteration and increment process.

- reviewing and selecting a CASE Tool,
- executing the experiment of four modules,
- measurement and evaluation of the results.

3. Improvement Activities

The UML is a synthesis of earlier object design languages. It includes a set of consistent diagrams, which are used to describe and communicate a software system's requirements, designs, and code. In this project we used the object oriented method with the UML for the project management and project development process. In order to achieve the goals of the PIE project we have performed the following activities:

• Selection of a CASE tool

A development environment consists of all the hardware and software required to develop the products efficiently. It includes choice of workstations, design tools, editors, compilers etc .

In the last time JAVA and its development environment have been rapidly improved. It is especially fit to apply for networking programming. With the aid of the review [4] we determined to purchase "JBuilder 3.0 Enterprise" from the company *Inprise* for the PIE project.

There are different CASE tools existing in European market. In order to choose to a suitable tool for the project, we reviewed firstly the various tool; then considering the requirements of the project, two CASE tools: Rational Rose and Together, were chosen for further test by using the modules of the PIE project.

When checking these two CASE Tools the following factors are considered:

User-Interface: A CASE tool is designed to assist in software development, and it should be logical and easy to use by designers without long training periods. A tool should support familiar GUI elements and UML, and integrate other applications such as Microsoft word, etc.

Workflow: Because of the weakness in the process of the analysis and the design phase, we regarded workflow as an important factor. A CASE Tool is able to model workflow of a new system in the field of the new system and to represent the software development process.

Code generation: On the basis of logical diagram a CASE tool must generate proper code in an appropriate format which implements the product model.

Reverse Engineering: Reverse engineering is the process of creating a model by analysing source code. As a software supplier, we offer our customers various products. It is often that we are asked to satisfy some special requirements by customers. A CASE tool has to allow to modify easily from analysis to design and to implement, and back to analysis again. The iterative style of development allows designers to begin with a set of known requirements, then evolve as project parameters change or new requirements are added and the project modelling is modified. The reverse Engineering and forward engineering are very important features for the dynamic development process, so that we alter the implement, assess the changes and incorporate them in the design.

The result of the test show that Rational Rose was the suitable tool for the PIE project and the application environment in Huengsberg AG [6].

- **Analysis and definition of the software development process**

The existing process of the software development consists of the definition, plan, function pattern, prototype, beta and series phase. This process orientates the requirements of the customers. We focused on checking and measuring the end products; we did not pay much attention to a permanent development methodology and the quality of the

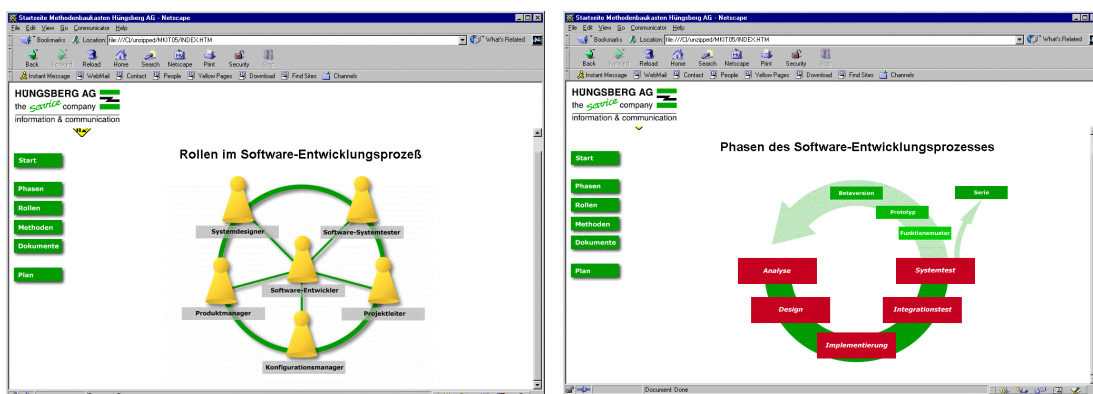


Fig. 3. The examples of the new defined development process. Left: the view of new defined roles in the development process; Right: the improved process of the software development.

development process. Moreover the requirements of the customers may be inadequate at the beginning of the project, and unstable during the process of the software development. Sometimes the customer's needs may change. In order to be successful in executing the project, the current process of the software development is not avail and must be improved, new tools should be employed for the process of the software development on the basis of the UML.

With the aid of a consulting company the current application environment is firstly analysed by using the new method kit. Some weak areas in the software development process have been discovered. Combining the existing environment with object-oriented development method, the process of the software development is defined carefully again and completed in detail.

The Fig. 3 shows the examples of roles in the software development process and the improved process of the software development.

Roles are defined as a set of related tasks of a person. In the process of the software development the six roles are defined. Each role has the responsibility to achieve the predefined objectives.

The improved process is made up of analysis, design, implementation, test and etc., 8 phases, which are based on the V-model; spiral model and current development environment.

The Tab. 1 presents the view of results of the definition of the development process for documents, roles, phases and methods.

| Dokumente | Beschreibung | Beispiel | Nutzungsanleitung | Checkliste | Phasen | Rollen | Methoden | Kommentar | | | | | | | | | | | | | | |
|-----------|---|----------|-------------------|------------|---------|--------|-----------------|------------------|------------|-----------------------|----------------|---------------------|----------------|---------------------|-------------------|--------------------------|---------------------|----------------|---------------|-----|--|---------------|
| | | | | | Analyse | Design | Implementierung | Integrationstest | Systemtest | Konfigurationsmanager | Projektmanager | Software-Entwickler | Systemdesigner | Änderungsmanagement | Change Management | Konfigurationsmanagement | Programmierschritte | Review-Technik | Testverfahren | UML | | |
| 2 | | | | | | | | | | | | | | | | | | | | | | |
| 3 | Pflichtenheft | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_pflich.htm |
| 4 | Functional Specification | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_funcsp.htm |
| 5 | Systemtestfallspezifikation | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_sysstf.htm |
| 6 | Projektplan | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_projpl.htm |
| 7 | Glossar | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_glossr.htm |
| 8 | Systemdesign | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_sysdes.htm |
| 9 | Benutzungshandbuch | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_benutz.htm |
| 10 | Integrationstestfallspezifikation | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_inttfs.htm |
| 11 | Programmschnittlinien | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_progri.htm |
| 12 | Moduldesign | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_moddes.htm |
| 13 | Quellcode | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_quellc.htm |
| 14 | Modultestfallspezifikation | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_modtfs.htm |
| 15 | Modultestprotokoll | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_modtpr.htm |
| 16 | Integrationstestprotokoll | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_inttpp.htm |
| 17 | Systemtestprotokoll | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_sysstpp.htm |
| 18 | Release Note | r | | | # | | | | | | | | | # | # | # | # | # | # | # | | d_relnot.htm |
| 19 | Beschreibung (I = Input, O = Output, M = Modifikation) V = verantwortlich, M = unter Mitarbeit von, # = Link, a = in Arbeit, r = in Review, ok = freigegeben) | Zustand | r | r | r | r | r | r | ok | ok | ok | ok | ok | r | r | r | r | r | r | r | | |
| | | | | | | | | | | | | | | | | | | | | | | Dateiname |

Tab. 1: The view of the definition of the documents, Roles, phases, and methods.

- **Training activities**

The team of the PIE project consists of the project leader, a software engineer and one contributor who assist in the development of special functions and programming in part time. In order to execute the PIE project an intensive program was scheduled for training team members.

Software developers took part in the seminars about the object-oriented software development, unified modelling language, design patterns and OO-programming and software metrics. They gained a lot of knowledge in the object-oriented analysis, design, implementing and metrics.

- **Execution of the experiment**

The UML consists of nine diagrams. In this project the Use Case diagram, Class and package diagrams, Sequence diagrams, Activity diagrams and Component diagrams have

been used for managing the project and for the process of the analysis, design and implementation. The following examples illustrate the application of the UML for the analysis, design and implementation.

Documenting the behaviour and requirements with UML: The first step in the development of a software products is to achieve a understanding of the problems and define the behaviour of the products. This begins with the assessment and documentation of the product requirements. In this project the use case model is used in documenting the behaviour and requirements of the project. The use case model illustrates the system's intended functions (use case), its surroundings (actors) and relationships between the use case and actors (use case diagrams). It provides a view of the system structure and one starting point for design. Fig. 4 shows the actors in the project and the use case diagram of the PIE project. It provides a detailed view of the project for the communication between the development team members and customers.

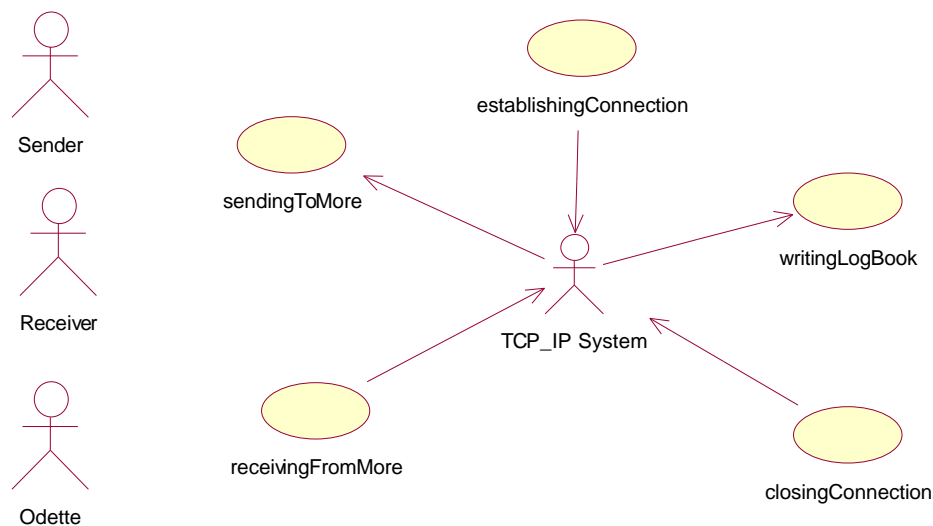


Fig. 4: The actors in the project and the use case diagram of the sender and receiver process, Logging activities and interprocess communication. This presents a view of the functions of the TCP/IP subsystem.

Software design with UML: A good software design must support the below abilities: Comprehensibility, maintainability and extensibility. In this project we employed the class and package diagram, sequence diagram, activity diagram and component diagram for the design. The Fig. 5 and Fig. 6 as instances display the class diagram and the sequence diagram.

The class diagram of three components in the project is shown in Fig. 5. The class diagram provides a static view of the classes in the logical view of the design , which illustrate how the classes relate. The class diagrams are also the foundation for code generation.

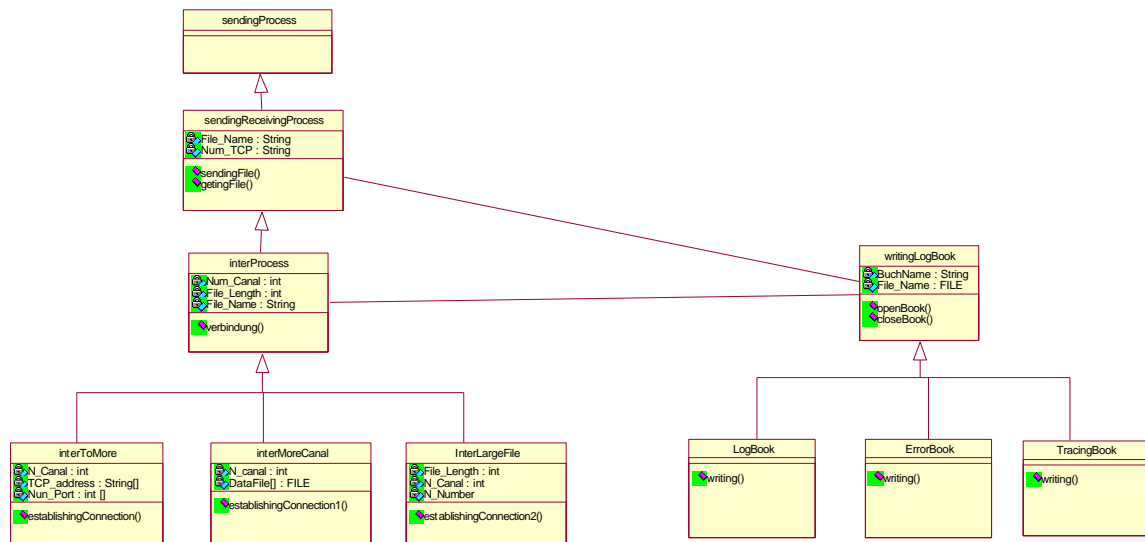


Fig. 5: Class diagram of the project components: sender and receiver process, logging activities and interprocess communication. It presents the view of the classes in the model and the interactions among them.

Class diagram and package diagrams are static. They are not adequate to determine whether the design is adequate to meet the requirements. From the class diagrams we cannot learn about the behaviours of the system. Sequence diagram can meet this need. The elements of sequence diagrams are objects and messages. The Fig. 6 shows the sequence diagram of the project DAX 2000 as example, where the boxes with underlying dashed line indicate objects.

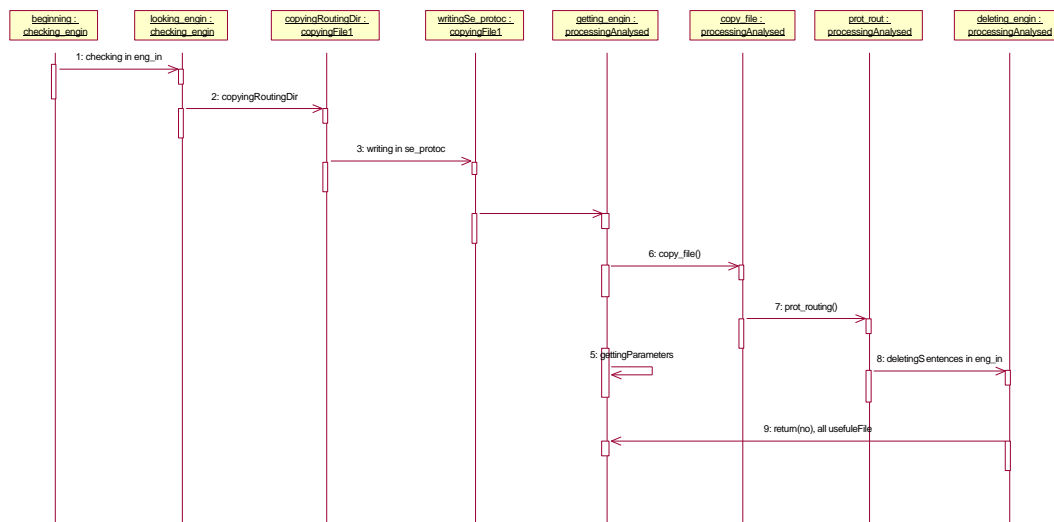


Fig. 6: Sequence diagram of process of the incoming file in DaxENGdat of the project DAX 2000. It shows the object interactions arranged in time sequence.

Sequence diagrams are derived from the development of use cases; they present the objects, messages and object interactions arranged in time sequence.

4. Results and further Aspects

In the project the object-oriented methods with UML are used for project development. During the execution of the experiment, we gathered the following information:

- The UML is a very useful language for modelling a software development process and enables developers easy to communicate with users of products and members in a development team. With the aid of UML and proper CASE tool, the software development process can flexibly and quickly be modified according to the new requirements of customers.
- By using the UML each step in the software development process can be represented in detail, so the quality of the development process can be easily examined and controlled.
- The object-oriented thinking is a foundation for a developer to use a UML properly and smoothly.
- After introducing the UML and CASE tool, the code of class level can be generated by class diagrams, the development period of the modules in the project is reduced.
- The CASE Tool „Rational Rose“ is especially suitable for modelling software development process. The generation of source code works properly and the reverse engineering has good functions. The iteration and increment development process can be more easily implemented by using the round-trip engineering.
- Consistency checking must be performed throughout the iteration and increment process and the life cycle of a project, because several views of the system are under development in parallel and care must be taken to ensure that models stay in synch.
- Combining UML with the new paradigm used to manage the software development, the software development process can be analysed and administered in a proper and efficient manner. Especially the weak areas of the software process in our company have been recognised, this is a basis for further improvement.
- The generated code consists of the class definitions. It is not the complete executable code. This must be implemented by hand, and so it is necessary to generate clear and concise code type for further programming.
- All of the diagrams in the UML presents the same model in varied views on the basis of different detailed aspects. Some diagrams are excessive diversity. The connection and dependence among diagrams have been neglected.

5. Conclusion

In this report we present the results of the OFTPIVE.PIE project. It is indicated that the introduction of UML and CASE Tool results in positive impact in software development process. Especially the UML is suitable to apply for modelling the software development process and for representing the analysis and design phases. This method has been used in modelling the project *DAX 2000* in the process of our software development. We will continue applying this method for the process of other software product developments.

6. References

1. H. Balzert: "Lehrbuch der Software-Technik", Spektrum Akademischer Verlag Heidelberg, Berlin Berlin, 1988.
2. A.Barthel, B. Hindel : "*The Method Kit: A new Paradigm for Process Definition*", CONQUEST' 98, pp.192-200, Sept. 1998 in Nuernberg, Germany.
3. W. Huengsberg: "*Ein Extranet im Internet für die Unternehmenskommunikation in der Automobilindustrie*", Sonderdruck des Beuth Verlags aus edi-change, No. 2 1997.
4. R. Gema: "*Gruppenbild, Java-Entwicklungsumgebungen im Praxisvergleich*", Magazin für computer technik, pp. 180-189, No. 5 1999.
- 5 B. Oestereich: "*Objekt-orientierte Softwareentwicklung*", R. Oldenbourg Verlag München Wien, 4. Auflage, 1998.
6. A. Goerges, M. Liu: "Tool Selection Review Report", the report of the ESSI project, July 1999.
6. M. Cantor: "Object-oriented Project Management with UML", Wiley Computer Publishing Toronto 1998.

Autors:

Arthur Görge is Chief Operating Officer Research & Development at Huengsberg AG. Since 1983 he has been working on computer technology, networking and electronic communication. His main areas of interest: project management of software development and the application of ISDN, internet technology for electronic communication in the automotive industry.

Minsheng Liu received the Ph.D in applied Physics at the University of Frankfurt am Main in 1998. He is working as a software developer at Huengsberg AG. His main interests are: object-oriented programming with JAVA; UML, CASE Tool, TCP/IP and the application for software development process.



Improving Quality in the Software Development Process on the Basis of UML

Arthur Görge and Minsheng Liu

HÜNGSBERG AG
the *service* company



A. Görge, M. Liu Hüingsberg AG
<http://www.daxware.com>

1. Executive Summary



In the PIE project the object-oriented method with UML and a new paradigm approach have been introduced to the application environment of our software development to improve the process of software development:

- Facilitating project management in proper and efficient manner,
- Discovering the weaknesses of the software development, redefining the process in detail and improving it; controlling the quality of each phase
- The number of the code with JAVA in components was reduced by 25 % compared to the number of C code.
- The automatic code generation can reduce the development time of the implementation of the project development.

Conclusion: Positive impact on and beneficial for the software development.

HÜNGSBERG AG
the *service* company



A. Görge, M. Liu Hüingsberg AG
<http://www.daxware.com>

2. Business Motivation:



The most important motivation of the PIE project:

- Improving product quality by reducing errors in analysis, design and implementation.
- Reducing the period of product development by employing efficient management and introducing new software development methods.
- Developing new products and improving the position in the market.

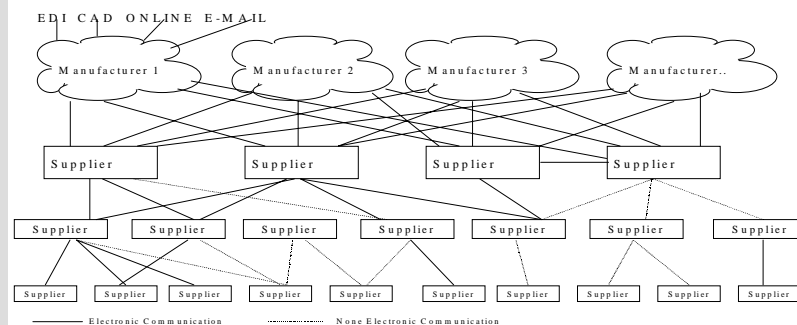


3. Description of the Experiment



- Background:** File transfer based on ODETTE: Point to point links on ISDN, this can no longer match customer's needs, as CAD file volume is increasing.

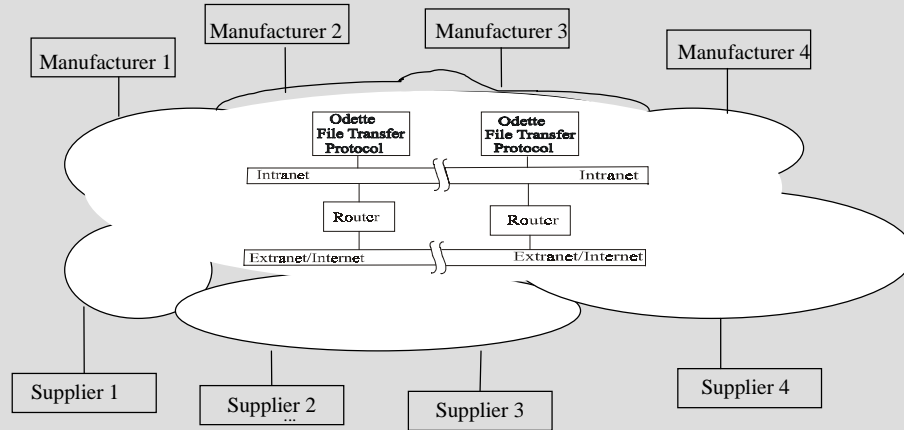
Today:



3. Description of the Experiment



□ **Aims and approach:** network for file transfer as follows:



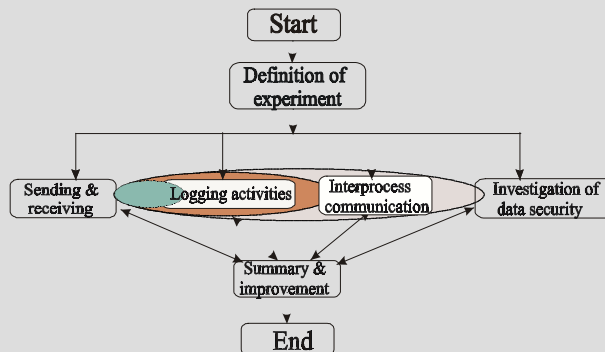
HÜNGSBERG AG
the *service* company

A. Görge, M. Liu Hüingsberg AG
<http://www.daxware.com>

3. Description of the Experiment



In the PIE project the object-oriented method with UML and the new paradigm approach have been used to realize four modules as follows:



Iteration and increment development process of the PIE project:

HÜNGSBERG AG
the *service* company

A. Görge, M. Liu Hüingsberg AG
<http://www.daxware.com>

4.Implementation Main Activities:



Main Activities:

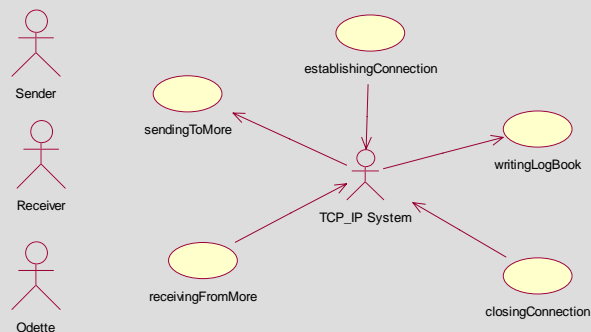
- Selection of the CASE Tool
- Improvement of the Software Development Process
- Training Activities
- Execution of the Experiment



4.Implementation Main Activities:



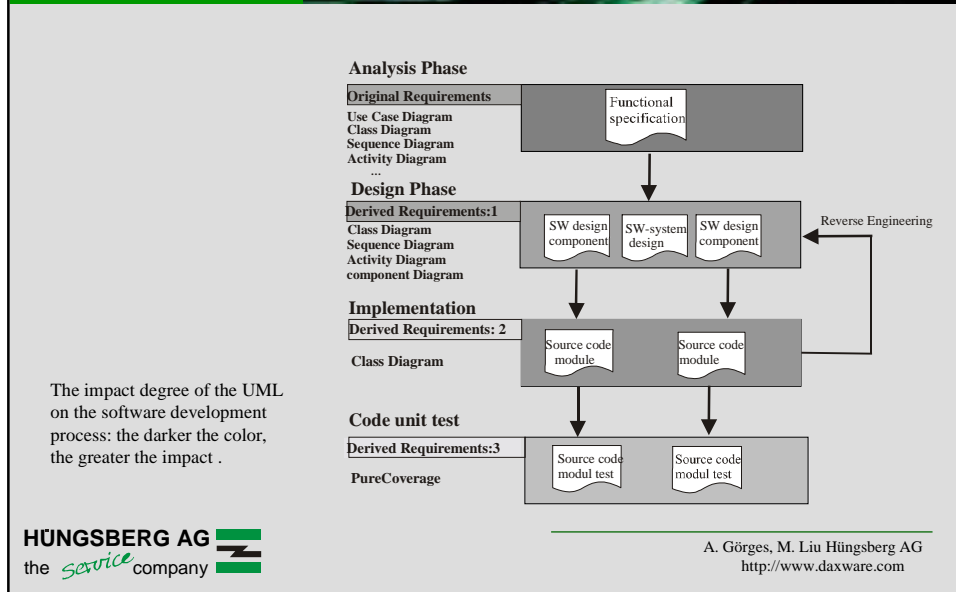
Execution of the Experiment: use case diagram as an example



The actors in the PIE project and the use case diagram of the modules.



5. Result and further Aspects



5. Result and further Aspects



During the execution of the experiment, we gathered the following information:

- The UML is a very useful language for modeling a software development process and enables developers easy to communicate with users of products and members in a development team.
- By using the UML each step in the software development process can be represented in detail, so the quality of the development process can be easily examined and controlled. **The iterative and incremental development process can be more easily implemented flexibly and quickly be modified by using the round-trip engineering with new requirements of customers.**
- The object-oriented thinking is a foundation for a developer to use a UML properly and smoothly.
- After introducing the UML and CASE tool, the code of class level can be generated by class diagrams, the development period of modules in the project can be reduced.
- Combining UML with the new paradigm used to manage the software development, the software development process can be analyzed and administered in a proper and efficient manner. Especially the weak areas of the software process in our company have been recognized, this is a basis for further improvement.

The generated code consists of the class definitions. It is not the complete executable code. This must be

HÜNGSBERG AG
the *service* company

A. Görge, M. Liu Hüingsberg AG
<http://www.daxware.com>

A Practical Approach to Validating and Testing Software Systems Using Scenarios

Johannes **Ryser** Martin **Glinz**
Department of Computer Science
University of Zurich
Winterthurerstrasse 190
CH-8057 Zurich, Switzerland
{ryser, glinz}@ifi.unizh.ch

Abstract. Scenarios (Use cases) are a means to capture a system's functionality and behavior in a user-centered perspective. Thus they are used in most modern object-oriented software development methods to help elicit and document user requirements. Scenarios also form a kind of abstract level test cases for the system under development. Yet they are seldom used to derive concrete system test cases. In this paper we present a procedure to use scenarios in a defined way to systematically derive test cases for system test. This is done by formalization of natural language scenarios into statecharts, annotation of statecharts with helpful information for test case creation/generation and by path traversal in the statecharts to determine concrete test cases.

1. Introduction

In developing a software system, validation and verification are recognized as vital activities. They are especially valuable when applied early in the development process, as errors found during the specification and design phase are much cheaper to correct than errors found in consequent phases [5]. Early validation and verification thus greatly reduce error fixing and fault cost.

Testing plays an important role in validating and verifying systems. Yet test preparation and the development of test cases is often done only just before testing starts, at the end of the development process, even though analysis as well as design would greatly profit from the insight gained by developers in creating test cases and preparing tests. Moreover, testing is often done in an ad-hoc manner, and test cases are quite often developed in an unstructured, non-systematic way. This is mainly due to the reality of commercial software development (only limited resources are available and only sparse resources are allocated to testing) and less to lack in available methods or lacking problem understanding. Any testing strategy has to address this practical issue if it is to be successfully applied. To improve testing in practice, systematic test case development and integration of test development methods with 'normal' system development methods is central. Test cases are only developed in a systematic way if clearly defined methods are applied. Test development methods will only be used if they are easy to apply, blend into existing development methods and do not impose an inappropriate overhead or intolerable cost.

Many strategies and approaches to testing exist. Besides established techniques like control and data flow testing or boundary analysis/domain testing [4, 17], formal languages for specification and specialized testing languages are gaining increased attention. Yet a gap is opening between the state of the art and the state of practice. The gap in-between what theoretically could be done and what really is done in practice, is mainly due to the following reasons (the list is not intended to be complete):

- **Lack in planning / time and cost pressure:** In real-world projects tests are conducted under immense time and cost pressure, as often the project at the end of the development process is behind schedule and over budget already. Detecting faults causes additional delays. As a consequence, both test preparation and execution are frequently performed only superficially. Cost and

time needed for testing are hard to be estimated with reasonable accuracy. Moreover, testing is often insufficiently planned for and not enough time and resources are allocated for testing.

- **Lacking (test) documentation:** Tests are not properly prepared, no test plans are developed and tests are not documented [29].
- **Drudgery:** Testing and test case development are tedious, wearisome, repetitious, error-prone and time-consuming activities which prompt fatigue and inattentive work, even if sound testing strategies and methods are applied.
- **Lacking tool support:** For this reason, testing has to be supported by tools. But only limited tool support does exist. Extended tool support and more especially automatic test case generation is restricted to systems which are formally specified. Even if automatic test case generation may be applied in a formally defined system, the resulting test suites are of immense size and generally only poor coverage is reached.
- **Formal languages /specific testing languages required:** Many test methods use formal specification languages or specific testing languages (thus requiring special training and education). Their application is extremely costly, they are difficult to apply and/or can only be applied to limited problems or very specific domains.
- **Lacking measures, measurements and data to quantify testing and evaluate test quality:** In most projects only little testing data (error statistics, coverage measurements, and so on) is collected during testing or available from other projects. Because of missing data only little can be said about the benefits and economics of testing, different approaches can not be compared and processes can hardly be improved. The quality of tests, and thus to some extent of the product, is often not assessed. Furthermore, the missing data further aggravates the problem of accurate test planning and allocation of the necessary resources.

The issues mentioned above may be addressed by various approaches. The problems of documentation and planning, for example, may be alleviated by improvements to the testing process, by use of and adherence to appropriate methods and clear definition of testing criteria, by testing strategies, or by a list of documents and deliverables that have to be produced during development of the system. Formal languages or specialized testing languages may allow for better automation of the testing process and for better tool support; closer integration of testing with established development methods may reduce cost and the need for special purpose languages, and (re)using software artifacts created in the analysis, specification and design phase may improve efficiency in test design and reduce drudgery and time pressure.

The strategy last mentioned above is the one pursued in our approach: To help bridge the gap between the state of the art and the state of practice, we propose the use of scenarios, not solely for requirements elicitation and specification (as done in leading object-oriented development methods), but for system testing, too, formalizing narrative natural language scenarios into more formal statecharts and deriving test cases from statecharts. We thus allow for – and enforce (in parts) – a systematic test case development. In designing the method, we try to utilize synergies between the phases of system analysis & specification and system test.

The rest of the paper is organized as follows: Section 2 serves as an introductory chapter to define and present the problem and shortly sketch the proposed solution. In section 3 we present the basic concepts and principles of the SCENT method, and describe the individual steps in the procedure of scenario creation, formalization and test case generation. In section 4 the method presented in this paper is compared to related work and in section 5 we present some conclusions.

2. Problem Disposition and Solution Strategy

In this section we take another look at some of the problems in testing and testing methodologies. Then we present the key concepts of the SCENT method and shortly introduce the notion of scenarios and use cases.

2.1. The Problem

To deliver a software product of high quality, an efficient, reliable quality process has to be implemented and sound (engineering) principles have to be followed and adhered to. And after all that can be done to construct quality products, testing as an analytical means to software quality has to be performed in a timely and systematic manner.

But in many projects, testing is done as a last minute effort to show the application to be functional and functioning, much more than to uncover errors and show its compliance to requirements. This is - at least partially - due to the following facts:

1. **Testing is done in the last phase of the development only:** Developers start the development of test cases only after most of the system development has been done. But testing can (and should) be started with as soon as the specification has been written. By developing test cases early in the development process, many errors, omissions, inconsistencies and even over-specifications may be found in the analysis or design phase still. It's cheaper to remove errors in the early phases.
2. **Testing methods are not integrated with (software) development methods.** Testing hardly uses any artifacts of earlier phases directly, but much work is needed to create test cases from the requirements specification and design models. It's easy to leave testing to be done at the end of the development, as testing and test preparation is not enforced earlier by the development methods.
3. **Test cases are not created/generated in a systematic manner.** Test cases are chosen randomly, by experience, according to some rules of thumb or according to insufficient criteria (statement coverage, input coverage, ...). Testers are left with no definite procedure on how to derive test cases.

These concerns can be reduced by extended tool support. But as mentioned before, testing is not a simple task that can be easily automated. It is not possible at the time being to automate the whole testing process and achieve acceptable test coverage in given time for projects relying on natural language specifications [23, 26]. Therefore, proper tool support helps to alleviate the problems mentioned above. It does not, however, solve them.

2.2. A Proposal to Solve the Problem: The SCENT Approach

We propose a practice-oriented scenario-based approach to support systematic test case development, that utilizes early artifacts of the development process in later phases again, in order to realize synergies between the closely related phases of system analysis and system test. We call our approach the SCENT method - A Method for SCENario-Based Validation and Test of Software.

In SCENT, we aim at providing a method that is - or easily can be - integrated with software development methods, a method that helps developers create test cases and think about testing early on in the development process and that supports systematic generation of test cases. SCENT enables scenario-based test case development for system test of software systems, taking, as is appropriate for system test, a functional testing strategy.

The key ideas in our approach are:

1. Use **natural language scenarios** not only to elicit and document requirements, to describe a system's functionality and specify a system's behavior, but also to validate the system under development while it is being developed,

2. **Uncover ambiguities, contradictions, omissions, impreciseness and vagueness** in natural language descriptions (as scenarios in SCENT are at first) by formalizing the narrative scenarios in statecharts [10],
3. **Annotate the narrative scenarios and/or the statecharts** where needed with pre- and post-conditions, data ranges and data values, and non-functional requirements, especially performance requirements, to supply all the information needed for testing and to make the statecharts suitable for the derivation of actual, concrete test cases,
4. **Systematically derive test cases** for system test by traversing paths in the statecharts and documenting the test cases.

These key concepts need to be supported by and integrated with the development method used to develop the application or the system, respectively. Most object-oriented methods support use cases and statecharts or comparable state-transition diagrams. Thus, the basic integration of the proposed method in any one of those methodologies is quite simple and straightforward.

In section 3 we describe the method in more detail.

2.3. Scenarios

Scenarios play an important role in our approach. But even though scenarios are nowadays ubiquitous and have long been used in human-computer-interaction, strategic planning and requirements engineering, a single, formal, agreed upon definition what a scenario is, does not exist.

We define scenarios informally to be any form of description or capture of user-system interaction sequences. The terms scenario, use case and actor are defined as follows:

Scenario – *An ordered set of interactions between partners, usually between a system and a set of actors external to the system. May comprise a concrete sequence of interaction steps (instance scenario) or a set of possible interaction steps (type scenario).*

Use case [13] – *A sequence of interactions between an actor (or actors) and a system triggered by a specific actor, which produces a result for an actor. A type scenario.*

Actor – *A role played by a user or an external system interacting with the system to be specified.*

3. Basic Principles of the SCENT- Method

By creating scenarios during requirements specification and system analysis, the requirements engineer produces a first set of abstract test cases. In SCENT, we reuse scenarios that were created during the analysis phase in test case development. The idea of using scenarios/use cases in testing is not new, however. Jacobson in his book and articles mentioned that use cases are well suited to be used as test cases for integration testing [13, 14]. Others have taken up, formalized and extended the notion of using scenarios to test a system (see for example [11] and [7]). In section 4 of this paper, a more detailed description of related work is given. But despite the ubiquity of scenario approaches, a practical method supporting testers in developing test cases from scenarios has not emerged yet [27]. The approach as presented in this paper represents ongoing research. The ideas presented are validated by application of the method in practice (see section 5).

The SCENT method comprises three main parts: Scenario creation, scenario formalization and test case derivation. All three are described in more detail in the following sections.

3.1. Scenario Creation

Many scenario processes are lacking a step procedure – a cookbook – for the creation and use of scenarios. In SCENT we define a procedure to elicit requirements and document them in scenarios. In this procedure we use a scenario template to format scenarios according to a common layout and structure. This template is not described in detail in this paper. A description of the template may be found in [24], the template itself may be downloaded from <http://www.ifi.unizh.ch/groups/req/ftp/SCENT/ScenarioTemplate.pdf>.

3.1.1. A Step Procedure for Scenario Creation

To create scenarios, first a list of all the persons and systems who interact with the system under consideration is created (or more precisely: a list of the roles these persons and systems play). All system in- and outputs are specified and all external events are listed. All the actors, the events and all system in- and outputs are uniquely named and a glossary of terms is created.

Having determined the actors, coarse scenarios are created capturing the main uses of the system. Ask questions like: “How does every actor interact with the system?”, “How does the system react to every external event?” in order to create short natural language descriptions of system usage.

These first scenarios might be on a type or on an instance level: They may describe interaction as seen by a distinct user (e.g. Fred Brown pushes the button) or on the more abstract level of roles (e.g. Fred Brown is an operator, thus: The operator pushes the button).

Table 1: Scenario Elicitation, Creation and Structuring

| # | Step Description | Results |
|----|--|---|
| 1 | Find all actors interacting with the system | List of actors |
| 2 | Find all (relevant system-external) events | List of events (triggers) |
| 3 | Determine results and output of the system | System output |
| 4 | Determine system boundaries | System boundaries |
| 5 | Create coarse overview scenarios (instance or type scenarios on business process or task level) | List of scenarios |
| 6 | Prioritize scenarios according to their importance and assure that the scenarios cover all system functionality | List of prioritized scenarios Links scenarios – actors |
| 7 | Transform instance to type scenarios. Create a step-by-step description of events and actions for each scenario (task level) | Coarse grained flow of actions in scenarios |
| 8 | Create an overview diagram | Overview Diagram |
| 9 | Have users review and comment on the scenarios and diagrams | Comments and annotations to scenarios |
| 10 | Extend the scenarios by refining the description of the normal flow of actions, break down tasks to single working steps | Description normal flow of actions Hints on test case derivation |
| 11 | Model alternative flows of actions, specify exceptions and how to react on exceptions. Include hints on test case derivation | Alternative flows of actions, exception handling in scenarios |
| 12 | Factor out abstract scenarios | Abstract scenarios |
| 13 | Include performance/ non-functional rqmts./ qualities in scenarios | Scenarios, annotated with qualities |
| 14 | Revise the overview diagram | Revised overview diagram |
| 15 | Have users check and validate the scenarios (Formal reviews) | Validated scenarios |

In the following steps, instance scenarios are transformed into type scenarios. The scenarios are refined by defining a step description for every scenario, and the scenarios are validated with the customer and/or the user. Alternative flows are modeled and abstract scenarios (sequences of interactions that appear in more than one scenario) are factored out.

Non-functional requirements and qualities are documented in natural language or with other appropriate means (formulas, timing constraints, pictures, graphics, screenshots, sketches, ...) in a special section of the scenario description. Abstract test cases are determined and information helpful to test case development is captured in the scenario descriptions (e.g. reminders what not to forget or what specifically to test for, values of particular interest, results of activities and computations that serve as an oracle in testing, ...).

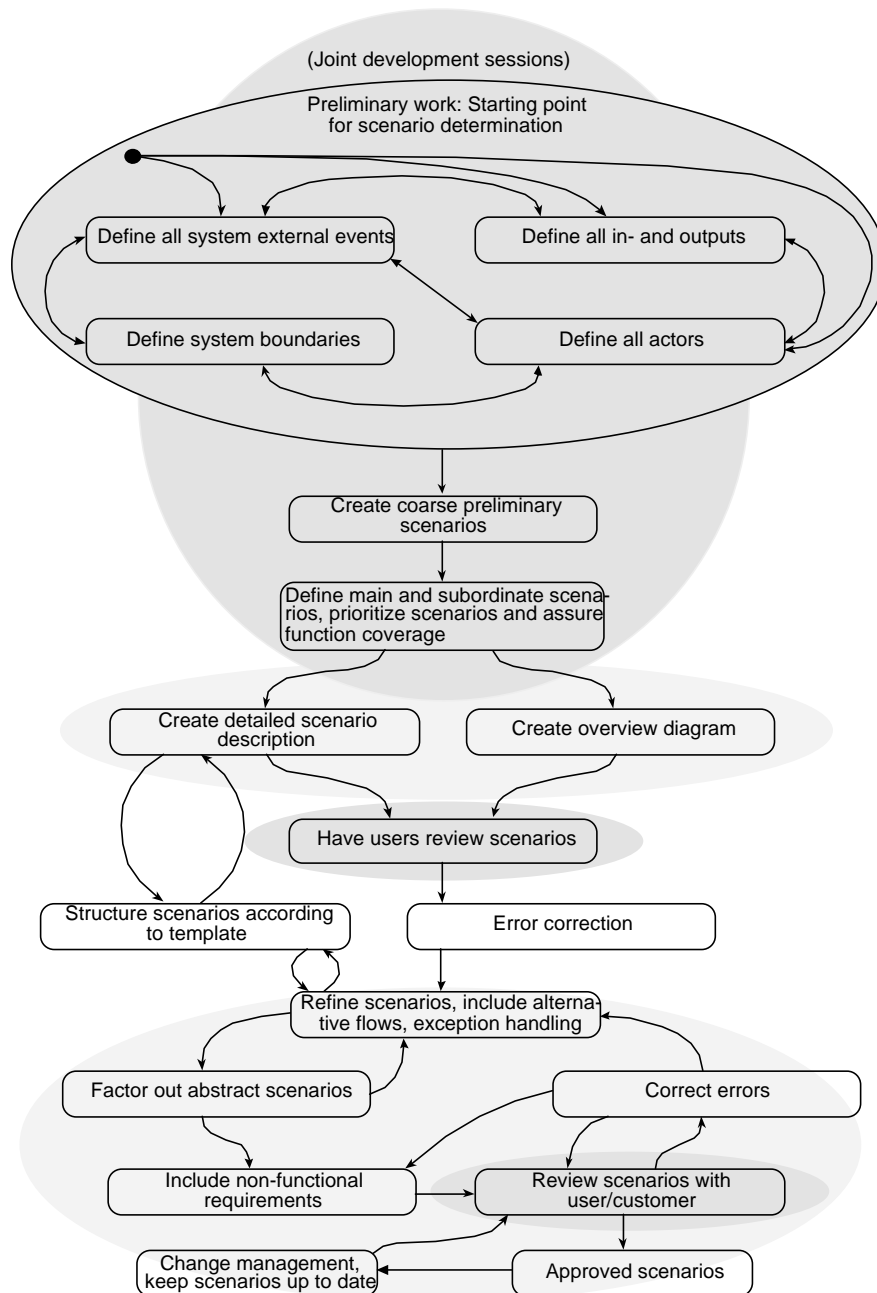


Figure 1: Scenario Elicitation, Scenario Creation and Structuring

Table 1 gives an overview of the 15 steps of the scenario creation process as defined in our method, and the purpose and results or deliverables of each step are listed.

Even though the procedure is presented as a sequence of steps, it is in reality highly iterative. Figure 1 shows the order of the activities in the scenario creation process. User involvement is depicted by shading: Darker shading indicates heavier user involvement.

3.1.2. An Example

As an example we choose the well-known and familiar automated teller machine (ATM). A short specification is given below (Figure 2).

At an ATM the customer may inquire the balance of his/her account or withdraw money up to a certain amount and at given piecing (only multiples of CHF 20 up to the personal limit may be dispensed). The customer needs a card and a personal identification number (PIN) to get access to the system and perform the mentioned banking transactions. The system interacts with a central bank system to get customer and account information and to inquire and update account balances. No receipts are issued.

Figure 2: The ATM machine specification

Because of space limitations only a short, partial description of the ATM example is given; the steps of the procedure are only touched upon to illustrate the procedure.

Below, excerpts of the scenario creation process for the ATM are presented. Numbers are relating to steps in the scenario creation procedure of Table 1.

1. Identify actors: In the example we identify four actors: the “*Customer*”, the “*Service Personnel*”, the “*Operator*” and the “*Banking system*”.
2. Identify external events: *Customer inserting card, entering PIN-code, choosing action, entering amount, taking back card, taking cash; operator filling bills; service personnel servicing machine.*
3. Determine system input, results and output of the system. System input: *Cards, PINs, choices for actions, amounts, bills.* System output/results: *Cards, balance info, cash/bills.*
4. Determine system boundaries: All persons and the banking system belong to the environment. Customer and account information are kept in the bank system.
5. Create coarse scenarios (instance or type scenarios on business process or task level): (1)*Inquire Balance*, (2)*Withdraw cash*, (3)*Service ATM*, (4)*Reload bills*
6. Prioritization of scenarios: First priority (1), (2), (4), secondary: (3). Assure that the scenarios cover all system functionality.

We further develop only one scenario. We choose scenario (2)*Withdraw cash*: A customer withdrawing money at the teller machine.

7. Create a step-by-step scenario description:

Scenario 2: Withdraw cash
 The customer withdraws money
 Actor: Customer
 Flow of actions:

1. The customer inserts the card
2. The system checks the card’s validity
3. The system displays the “Enter PIN” Dialog
4. The customer enters his PIN
5. The system checks the PIN
6. The system displays the main menu
7. The customer chooses “Cash Withdrawal” from the main menu
8. The system displays the cash withdrawal dialog
9. The customer enters the desired amount

- | |
|---|
| <ol style="list-style-type: none"> 10. The system returns the card 11. The system dispenses the money 12. The system displays the welcome screen |
|---|

8. The overview diagram is omitted to keep the example short. It is a “standard” use case diagram in UML notation.
9. Scenario validation: Have users review and comment on the scenarios and the overview diagram. Validation of the narrative scenarios in a first step is done by walking customers and users through the scenarios. Later on formal user reviews are scheduled and conducted (Step 15).
10. Scenario refinement: The steps in the coarse-grained scenario are refined to single, in the context “atomic” actions. The first step does not have to be clarified as it presents a single action by the customer. The second step may well be refined:

- | |
|--|
| <ol style="list-style-type: none"> 2.1 The system reads the card number and transfers the card number to the bank system to be validated 2.2 The bank system checks the card number and returns a validation code: Code 1: Card is valid, Code 0: Card not valid, return card to customer, Code -1: Card missing or reported as stolen, withdraw card 3. The system displays the “Enter PIN” Dialog 4.1 The customer pushes a numeric key 4.2 The system displays a masking character (echo key) in the input field on the screen 4.3 The customer pushes a numeric key 4.4 ... |
|--|

11. Model alternative flows of actions, specify exceptions and how to react to them. In SCENT, exceptional flows are separated from the normal flow of actions. By doing so, the developer of a scenario is forced to consciously think about alternatives and exceptions that could happen in any and every single scenario step. Moreover the normal flow of actions thus remains uncluttered by alternatives.

In the example above, step 1

- | |
|--|
| <ol style="list-style-type: none"> 1. The customer inserts the card |
|--|

may have the exception that the card can not be entered (e.g. the slot is obstructed). The corresponding entry in the alternatives section of the scenario description may read:

- | |
|--|
| <ol style="list-style-type: none"> 1a. The slot is obstructed <ol style="list-style-type: none"> 1a.1 The customer informs a human teller or calls and informs the service department. 1a.2 If a human teller was informed: The teller informs the service department 1a.3 The service department repairs the machine. Goto scenario (4)Service ATM |
|--|

12. Factoring out abstract scenarios: Certain sequences in a scenario might be reused in other scenarios. In the example, the authentication procedure is factored out:

- | |
|---|
| <ol style="list-style-type: none"> 1. The customer inserts the card 2. The system checks the card’s validity 3. The system displays the “Enter PIN” Dialog 4. The customer enters his PIN 5. The system checks the PIN 6. The system displays the main menu |
|---|

13. Include non-functional requirements in scenarios: Performance requirements are included in the scenarios, qualities are appended to the scenarios. As an example, we assume that the validity-check of a card that has been inserted at the ATM must be performed in less than two seconds. Moreover, the color red is only to be used for error messages. The second step in the scenario description is correspondingly changed to read:

- | |
|---|
| <ol style="list-style-type: none"> 1. The customer inserts the card 2. The system checks the card’s validity. This operation must take less than two seconds 3. The system ... |
|---|

and the requirement on the use of the color red is appended to the scenario:

...

12. The system displays the welcome screen

Non-functional Requirements: The color red is to be used for error messages only

14. Revise the overview diagram: Abstract scenarios, newly found scenarios and scenarios that have been divided or joined have to be updated in the overview diagram. The diagram and the scenario descriptions have to be kept consistent.
15. Scenario validation: Have users check and validate the refined scenarios (Reviews). Scenarios are altered and updated according to errors and problems found (Iterate through steps 10 to 15 of the procedure).

3.2. Scenario Formalization

Scenarios are validated by users and customers throughout the scenario creation process by reviews (inspections) and walkthroughs (see Section 3.1.). Scenarios prove valuable in validating requirements: As (functional) requirements are captured in the form of interaction descriptions, the user does not have to read and validate an enumeration of required features, abstract functions and qualities that are pulled out of usage context (as they are in traditional specifications). Requirements are captured in descriptions of the flow of actions. Thus, scenarios ‘naturally’ bundle requirements that belong together. They do so longitudinally, that is from the start of a transaction to the end of transaction. Dependencies between requirements and the interactions between features are at least partially described in scenarios as well.

Yet natural language scenarios, as they have been created in the scenario creation process so far, suffer from the problem of all natural language specifications: Natural language is not precise, definite and unequivocal (as is shown by this very sentence: Does the negation of ‘not precise’ also extend to ‘definite’ and ‘unequivocal’, or is the scope of the negation limited to the adjective directly following it?). Narrative scenarios may be ambiguous, inconsistent and incomplete. Reviews by users may find some of these problems, but many inconsistencies and omissions might slip by undetected.

Formalization helps in finding and avoiding these problems. Formal languages allow for formal reasoning, (strong) verification and proof of correctness. But formal languages have their own shortcomings, too: They require knowledge of a special language, are hard to understand and their application may be error-prone.

In SCENT we take an intermediate way by converting natural language scenarios into semiformal statecharts. This formalization helps to find many omissions, ambiguities and inconsistencies, yet the graphical representation of scenarios can well be understood by users, given some guidance by the developers. Thus, the formalization is a very helpful fault-finding procedure and can be seen as a part of static testing.

3.2.1. The Formalization Step

In the formalization step, structured natural language scenarios are transformed into statecharts. This step is in SCENT informal itself. In fact, the mapping of scenarios to statecharts is a creative modeling step that can not be formalized. The statecharts created from scenarios by one developer might significantly differ from statecharts developed from the same scenarios by another developer.

We define some heuristics to support developers in the transformation step. The heuristics are:

- As soon as first scenarios have been developed (step 5 and 7 in the scenario creation procedure), create statecharts to match the scenarios.
- Create a statechart for each scenario. The normal flow, all exceptional flows and all alternatives of a given scenario are captured in one statechart.
- The statecharts are refined along with the scenarios, thus providing for a continuous validation and ongoing check for inconsistencies, omissions and ambiguities in the narrative scenarios. As the coarse overview scenarios are refined to reflect the interactions on task level, new states are

introduced in the statecharts, states are expanded to comprise substates, and parallelism may be caught in parallel states, as appropriate.

- Model the normal flow of a scenario first. Integrate the alternative flows later on. Check if alternatives are missing. This can be done by investigating which events may occur in a given state. If an event could occur that has not been modeled, a transition is missing and usually an action or an alternative flow in a scenario is missing as well.
- Represent abstract scenarios as hierarchical statecharts.
- A single step in a scenario usually translates into a state or a transition in a statechart¹. As the steps are mapped to either states or transitions, missing states and transitions will emerge and need to be added. Superfluous states (and transitions, if any) need to be deleted or merged with needed states (or transitions).
- External and internal events are mapped to state transitions, the triggering event is the state transition to the initial state.
- At first, statecharts are not integrated. These partial models help to enable traceability (from design and tests to requirements and vice versa). Statecharts may be integrated later on to create a model of the full system [8].
- Check the statecharts for internal completeness and consistency. Are all the necessary states specified? Are all the states in a statechart connected? Has every statechart an entry and an exit node? Are there no dangling links? Can every state be entered and left (except the final state)? Are all the necessary transitions specified? On what events will a state be entered, on what events can a state be left? Check the event list created for scenario elicitation to see if all relevant events are handled. Ask questions like: “The system being in this state, what will happen if the user does this or that?” “Are states and events named expressively and consistently, following some scheme?”
- Cross-check statecharts. Do states, transitions and events appearing in more than one scenario have the same names?

Creation of scenarios and of statecharts is an iterative process. Statecharts have to be validated with the user either by inspection or review, or by paraphrasing and recounting them to the customer (end user, procurer) in a narrative style. All (important) paths are traversed; the developer guides the customer through the flows. This validation activity works hand in hand with the phase of test case derivation: The paths traversed with the customer to validate the statecharts are test cases that need to be tested in system test. Again it has to be emphasized that the process is not a sequential one, many of the activities may – at least partially – be done in parallel; they profit one from another and make use of the same artifacts.

The statecharts developed at first pass may be integrated to a full system model in a following step (as desired. It is not mandatory in the method, but might help in design as the integration of all statecharts represents a full system model).

3.2.2. Statechart Annotation

Statecharts describe the behavior of a system (how does a system behave in response to events and given conditions, ...), but let out other important information as data, performance and qualities. Nevertheless, this additional information is important for testing: Many errors are data-related and may only be found by test cases that can not be directly derived from statecharts (The sample bug

¹ If a working step in a scenario does not map to a state or transition, but needs to be modeled in more than one state or transition, respectively, this usually indicates that the step should be refined (broken down into substeps, its components). This will not normally be the case, as states can be created at various levels of abstraction and at various granularity levels - to the modelers will. But states at quite different abstraction levels in one statechart are an indication for insufficiently subdivided and refined scenario steps.

statistics in [3] show that this kind of bugs might well account for up to one third of all errors). Moreover, concrete test data, that is input values and expected output, can only partially be derived from statecharts directly.

For this reason, we extend the statecharts notation to include information important for testing. In particular, the additional testing information may comprise the following:

- Preconditions (and postconditions as needed)
- Data: Input, expected output and ranges
- Nonfunctional requirements.

The information is captured in annotations as shown in Figure 4.

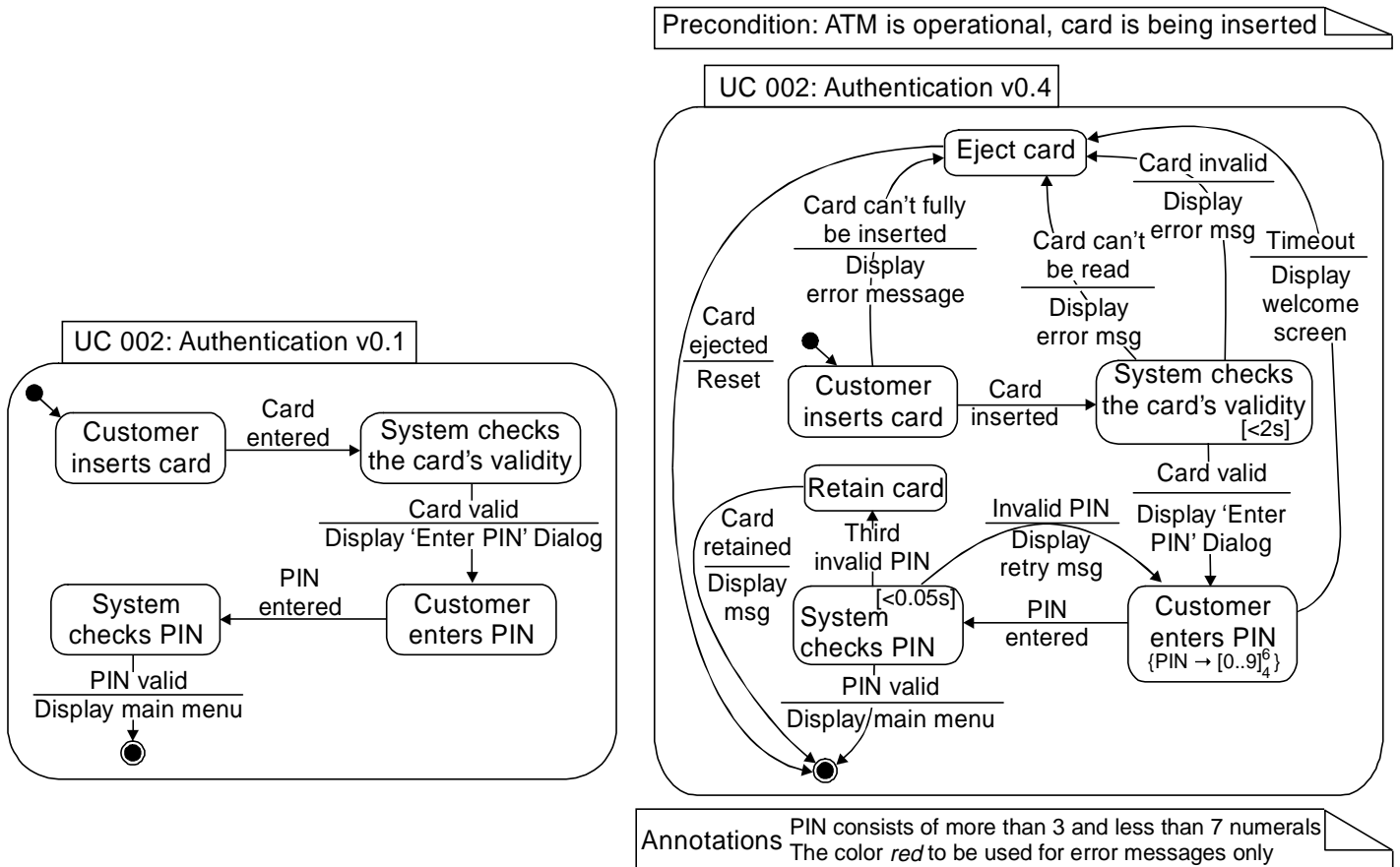


Figure 3: A statechart representing the ‘Authentication’ scenario

Figure 4: ‘Authentication’-Statechart with alternative flows and annotations

Preconditions are captured in banner-like notes. Data is annotated to states and – if applicable – to transitions alike: Ranges are specified in square brackets (or alternatively in curly braces, if square brackets are used for other purposes – see Figure 4 for an example). Expected results can be specified for key-values. Descriptions of expected data formats and restrictions may be specified in the statechart and/or references to the specification may be inserted. If desirable (for readability or understandability of the model, for example, or if paralleled in the domain), specific states for data validation may be modeled.

Performance requirements are annotated in square brackets as well. Performance requirements spanning more than one state or transition are specified by attaching the (timing) constraint to a dashed line connected to the affected states/transitions.

3.2.3. An Example

As an example of the formalization process, we consider the abstract scenario “Authentication” of the ATM example introduced in section 3.1.2, step 12.

At first the normal flow of actions is modeled in a statechart (see Figure 3).

Then the alternative flows are modeled (see Figure 4).

Once a scenario is modeled in a statechart, the statechart is annotated as needed. Preconditions and data annotations are included in the statecharts. In the example, valid and invalid PINs are distinguished by state transitions, but no indication as to what an invalid PIN is, is made. So a data annotation may specify the ranges and the form of a PIN (see Figure 4). Furthermore, two performance constraints have been specified in the example: The verification of the card’s validity shall not take more than two seconds (assuming the network connection in-between ATM and banking system to be sufficiently fast) and the validation of the PIN shall not take more than five hundredth of a second (assuming the PIN can be validated algorithmically, just knowing the PIN and an encryption key read in from the card).

3.3. Test Case Derivation

Test case derivation in the SCENT method comprises three steps:

Step 1 (mandatory). Test case derivation from statecharts.

Step 2 (mandatory). Testing dependencies among scenarios and additional tests (e.g. testing for specific qualities).

Step 3 (optional). Statechart integration and test case derivation from the integrated statechart

In this paper, only the first step is described in more detail. To support the second step, we define a new diagram type called dependency charts in SCENT. In dependency charts, timing, logical and causal dependencies between scenarios are captured and depicted. Thus, testing dependencies among scenarios is supported by test case derivation from dependency charts. For a detailed description of dependency charts see [24]. Additional test cases are developed using special testing information supplied in the scenarios: The notes taken during scenario creation and refinement now are used to enhance the initial test suite.

3.3.1. Test Case Derivation from Statecharts

In SCENT, test cases are derived by path traversal in statecharts. First, the normal flow of actions represented in the statechart is followed, then the paths representing the alternative flows of actions and the exceptions are traversed. In the method, we cover all nodes and all links in the graph, that is: all states and all transitions are covered by at least one test case. If desired, a more elaborate coverage could be chosen (e.g. switch or n-switch coverage [6, 9, 18]). Most states and many transitions are traversed more than once as annotations are used to refine test cases. Abstract scenarios are integrated in the calling scenarios, to allow for thorough tests of single scenarios. The statecharts are not integrated, though. The partial character of scenarios is thereby preserved, enabling traceability from test cases to requirements and vice versa. Furthermore, the user-oriented view of scenarios is thus promoted into testing, scenario prioritization may be utilized to determine test priorities, and finally the state-space of the (partial) solutions is thereby limited to prevent a combinatorial state/transition explosion. The incremental development procedure as encouraged by the use of scenarios is supported by not integrating the statecharts: Changes in a scenario (=changes in system usage) are usually confined to changes in one statechart and changes to test cases derived from the one statechart.

Annotations in the statecharts are taken into account in developing tests: Preconditions to statecharts define test preparation that has to be done before test cases derived from the statechart can be executed – the testing setup is determined by the preconditions.

The data specified in the scenarios and annotated in the statecharts help develop boundary value tests – tests that traverse the same path in the statecharts for every boundary value of data ranges as well as for a data value just above and/or below the boundary, as is done in boundary-value analysis. Domain testing techniques and data flow testing can be applied to derive further test cases ([3, 4, 17]). Furthermore, as path traversal in statecharts will only generate tests for valid sequences of events, the tester has to ensure inclusion of invalid event sequences in the test. These tests are constructed by evoking or generating events while the machine is in a state where it should not respond to the generated events. Thus, exception testing can be improved and systematized and a better test coverage can be achieved.

Notes on performance and nonfunctional requirements in the scenarios help to formulate test cases to test these properties.

3.3.2. Test Case Derivation in the Example

To illustrate test case derivation from statecharts, we present some test cases as created by path traversal of the statechart depicted in Figure 4. The first test case follows the normal flow of actions: The card can be inserted and the card as well as the PIN are valid. The next test case considers the exception of an incorrect PIN entered. Next an invalid PIN is entered (PIN too short; this test case takes into account the data annotations specified in the statechart). Finally, a third invalid PIN is entered to provoke another validation failure and traverse the *Third invalid PIN* link (see Table 2).

In the statechart, the paths that have been traversed are marked. If the developer/tester encounters a data annotation, be it on a link or in a state, he/she creates a test case for every boundary value/one above/one below. This means that in the example, the tester has to develop a test case using a key that is too short (three numerals), a key that is four and six numerals long, respectively, and a key that is too long (seven numerals). If a character key was used, the tester would test for keys using inadmissible characters. In this example it is obvious that state-transition tests are not sufficient: Even though the state-machine will differentiate between valid and invalid PINs, it does not indicate why a key is invalid. Is it because the PIN does not meet required syntactical or formal attributes (length, only admissible characters), or is it because the user has entered a PIN that is syntactically correct but not valid as it is not the user’s PIN? And even if this difference is modeled by distinct transitions (incorrect PIN vs. invalid PIN), still the tester has to test for all kinds of (syntactically) incorrect entries, e.g. key too long, key too short, key has inadmissible character, key does not include at least one non-alphanumerical character, and so on.

Annotations of performance requirements are tested for in like manner.

Table 2: Test Cases for the ATM Example

| <i>Test preparation:</i> | | ATM operational, card and PIN (1234) have been issued, card is being inserted | |
|--------------------------|--------------|---|------------------------------|
| <i>ID</i> | <i>State</i> | <i>Input/User actions/ Conditions</i> | <i>Expected output</i> |
| 1.1 | Card sensed | Card can be read, card valid, valid PIN (1234) entered in time | Main menu displayed |
| 1.2 | Card sensed | Card can be read, card valid, invalid PIN (1245) entered in time (first try) | Retry message displayed |
| 1.3 | Retry msg | Invalid PIN (123) entered in time, second try | Retry message displayed |
| 1.4 | Retry msg | Invalid PIN (1234567) entered in time, third try | Card retained, user informed |
| ... | ... | ... | ... |

The test cases in Table 2 might well be refined to reflect more details in requirements: The normal flow of actions captured in test case 1.1 in Table 2 above can be broken down to single steps as illustrated in Table 3. The decision on the level of abstraction in testing (to what detail shall be tested) depends on the kind of test (unit level, integration, system) to be run and on the testing that has been performed before (full tests on a system level can not be done because of the resulting test suite size or at least is not economical to do so).

Table 3: Refined Test Cases for the ATM Example

| <i>Test preparation:</i> | | ATM operational, card and PIN (1234) have been issued, card is being inserted | |
|--------------------------|---------------|---|---|
| <i>ID</i> | <i>State</i> | <i>Input/Actions/ Conditions</i> | <i>Expected output</i> |
| 1.1 | Card sensed | Card is taken in | Card inserted, validation screen displayed |
| 1.2 | Card inserted | Validate Card | Card is valid, 'Enter PIN'-dialog displayed |
| 1.3 | Card valid | Customer enters PIN | PIN (1234) entered in time, validation screen displayed |
| 1.4 | PIN entered | Validate PIN | PIN is valid, main menu displayed |
| ... | ... | ... | ... |

4. Related Work

Even though literature on scenarios abounds, test case derivation from scenarios is yet in its infancy. Scenarios are used in most object-oriented development methods, notably also in the UML (Unified Modeling Language), and many different approaches have been developed over the last couple of years. Yet, in the area of testing, only few scenario-based approaches exist. In the following, we review some approaches with regard to use case creation, scenario formalization and support for testing activities:

- Jacobson's Use Case Approach [13-15] was one of the first to disseminate the use of scenarios and propagate a user-centered requirements capture and specification. The approach does not, however, propose any defined procedure on how to create scenarios, nor on how to use scenarios in testing. Scenarios are not formalized in the approach. No specific description format or template is advocated. Furthermore, use cases – as groupings or collections of functionalities and requirements – are rarely truly independent in practice. But in Jacobson's work only very limited support for modeling dependencies between scenarios is given (*uses* and *extends* relations). However, to derive tests from scenarios the dependencies between scenarios have to be known, else crucial parts of the system will/may not be tested for.
- Hsia et al. [11] have proposed and described a more formalized approach to scenario creation and validation, constructing scenario trees. Their approach is based on regular grammars and equivalent (conceptual) state machines. Scenario creation and formalization are core-parts of the approach. The use of scenarios to derive test cases however is only shortly sketched and no further procedure has been specified. Their main use of scenarios in testing is for acceptance test. Our approach is close to the one proposed by Hsia in many respects. It differs though in central issues:
 - Scenario elicitation is conducted via scenario trees in the Hsia approach. In our approach we define an iterative step-by-step procedure to create scenarios in structured natural language.

- Scenario trees are formalized into regular grammars in the Hsia approach. These grammars describe a conceptual state machine, defining a formal abstract model. One model for each user view is created. The definition, use and maintenance of these grammars is labor-intensive and requires special training and skills on the side of the developers. Furthermore the grammars are not intelligible to customers and users. Changes to scenarios reflected in the grammars are quite cumbersome.
In contrast we formalize scenarios into statecharts, every statechart representing one scenario. Statecharts need not be (but may be!) integrated. The notation is expressive and understandable to users, and changes in scenarios are easily reflected in the statechart representation.
- In the SCENT method a definite procedure for test case derivation from statecharts is specified, creating concrete test cases (defining the settings/environment and the input values needed for test execution). In the Hsia approach, basis paths are used to generate scenarios from the conceptual state machine, these scenarios are used as input for acceptance testing. No concrete test cases are created.
- In our approach, statecharts are annotated with preconditions, data and nonfunctional requirements to enhance the creation of test cases. No equivalent concept has been defined in the Hsia approach.
- Dependencies and relationships among scenarios may in SCENT be modeled in dependency charts [24] and dependencies may be tested for accordingly. In their approach, Hsia et al. do not propose any way to handle inter-scenario dependencies.
- Firesmith [7] extends the scenario approach to model scenario lifecycles and the relationship between different scenarios. The benefit of scenarios in testing is only hinted at, the development of test cases is not addressed at all.
- Regnell et al. [20, 21] in their approach aim at overcoming the problem of lacking synthesis of single scenarios to reach a full picture of the whole system by formalizing and integrating the use cases of a system. Testing is touched upon, but no strategy to test case selection is defined. The main aim of [20] is to present improvements to the OOSE/Use Case Driven Analysis UCDA approach of Jacobson [13] by identifying weaknesses and problems in UCDA and defining a possible solution. [21] focuses on the representation, extending the former approach to include a hierarchical structured representation. Testing and the derivation of test cases is not an issue in the approach.
- Potts et al. [19] describe a scenario analysis approach with an emphasis on an inquiry-based procedure in the analysis process. They define a process for capturing and describing reasoning and discussion about requirements. In their approach, they take changing requirements and the reasoning process into account, supplying a model for scenario evolution. Testing as such is not an issue in their work.
- Lee et al. [16] use Petri nets for the analysis and integration of use cases. They emphasize the importance of incremental specification of partial system behavior and of consistency and completeness checks for requirements engineering techniques. Then they argue that an extended Petri net approach satisfies these demands. They define Constraints-based Modular Petri Nets (CMPNs), introduce Petri net slices to analyze system behavior, present a procedure to create CMPNs from scenarios and show how the model can be checked for consistency and completeness. As some of the approaches mentioned above, this approach tries to integrate and analyze use cases. However, it does not define a procedure for test case derivation from scenarios. No scenario creation procedure is specified either; scenario descriptions are input to the method.
- Message sequence charts MSCs were used in different approaches to formalize scenarios or to capture requirements of systems (see [1] for an example). Yet MSCs suffer from the disadvantage that they are getting overloaded fast when all exceptions and alternatives of a scenario are to

be integrated in one chart. On the other hand, they miss an abstraction mechanism to decompose complex system descriptions.

5. Conclusions

In this paper we have presented the SCENT method, a scenario-based approach to support the tester of a software system in systematically developing test cases. This is done by eliciting and documenting requirements in natural language scenarios, using a template to structure the scenarios. Scenarios then are formalized into statecharts. Finally, test cases are derived by path traversal in statecharts.

The method introduced in this paper has been applied in practice to two projects at the ABB Research Center in Baden/Switzerland. First experiences are quite promising as the main goal of the method, namely to supply test developers with a practical and systematic way to derive a first set of test cases, has been reached. The projects in which the method was applied were applications to remote monitoring of embedded systems [12].

The use of scenarios was perceived by the developers as helpful and valuable in modeling user interaction with the system. Developers especially appreciated the integration of the users and the direct feedback they received in creating and refining scenarios.

Not surprising, the creation of coarse overview scenarios and the iterative refinement of scenarios down to sequences of atomic actions proved to be valuable and was very much appreciated by users and developers.

The application of the method (and of scenario approaches in general) was not without difficulties, however. Some of the problems that surfaced in applying the methods were:

- Contrary to the positive experience in modeling user interaction in scenarios, it was troublesome to model system internal interaction and the interaction with other systems in scenarios as well. Instead of using scenarios, the developers preferred to create state machines and/or other models directly.
- Developers did not want to specify scenarios down to the level needed for test case design. Scenarios were used to model user interaction with the system at an abstract level, then refined to cover all the systems tasks. At this point it was not easy to convince the developers that further refinement and specification was needed, if scenarios were to be used for test case generation.
- Scenario management was considered a major problem throughout the development. As the primary scenarios created in SCENT are natural language descriptions and as they interrelate with many of the other artifacts produced in the software engineering process, there is threefold to manage: Keep scenarios consistent in themselves (as one specific scenario may exist in many versions and representations, e.g. as a natural language scenario and as a statechart, and as different scenarios may be related one to another), keep scenarios and other documents and artifacts consistent (e.g. all the links between scenarios and models derived from scenarios like class and behavioral models, and more especially the links between scenarios and other parts of the specification), and finally keep scenarios up-to-date when requirements, the environment and the developers' understanding of the problem are changing.

The formalization process posed some specific problems, as the mapping of scenario actions to states or transitions is not definite and clear-cut. A scenario transformed into a statechart by one developer may differ significantly from a statechart developed from the same scenario by another developer. Scenario formalization is not free, some extra work is needed; in fact, the development of statecharts might take quite some time and bind some resources. However, the extra work put into scenario formalization pays back in many ways:

1. As mentioned before, the transformation of structured-text scenarios into a semiformal statechart representation helps in verifying and validating narrative scenarios. Omissions, inconsistencies and ambiguities are found. The specification is thus improved.
2. Developers gain a deeper understanding of the domain and the system to be built because they have to understand the details to formalize the scenarios.
3. The statecharts created in the transformation may well be used and reused in design and implementation.
4. The formalized scenarios are (re)used in testing. Test case preparation and expenses are moved from the testing phase late in the development process to earlier activities, thus alleviating the problem of testing poorly done under time pressure. By using a systematical way to develop test cases, test coverage is improved.

The cost of developing the statecharts is justified by the benefits of an improved specification and enhanced testing.

Test case creation on the other hand was unproblematic (as expected).

Future work will be done in the direction of defining a coverage criterion for requirements captured in scenarios or in other natural language documents and descriptions, and measuring/quantifying the improvement in coverage gained by applying the SCENT method. We are also aiming at more tightly integrating non-functional requirements into scenarios and statecharts.

References

- [1] M. Andersson, J. Bergstrand, "Formalizing Use Cases with Message Sequence Charts," in *Lund Institute of Technology*: Lund, 1995.
- [2] M. Arnold, M. Erdmann, M. Glinz, P. Haumer, R. Knoll, B. Paech, K. Pohl, J. Ryser, R. Studer, K. Weidenhaupt, "Survey on the Scenario Use in Twelve Selected Industrial Projects," GI Fachgruppe 2.1.6 Requirements Engineering, *Aachener Informatik-Berichte* 98-07, June 1998.
- [3] B. Beizer, *Software Testing Techniques*, Second Edition ed. New York: Van Nostrand Reinhold, 1990.
- [4] B. Beizer, *Black-Box Testing, Techniques for Functional Testing of Software and Systems*. New York: John Wiley & Sons, 1995.
- [5] B. Boehm, *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice Hall, 1981.
- [6] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines," *IEEE Transactions on Software Engineering*, vol. 4, # 3, pp. 178-187, 1978.
- [7] D. C. Firesmith, "Modeling the Dynamic Behavior of Systems, Mechanisms and Classes with Scenarios," *Report on Object Analysis and Design*, vol. 1, # 2, pp. 32-36,47, 1994.
- [8] M. Glinz, "An Integrated Formal Model of Scenarios Based on Statecharts", in W.Schäfer, and P.Botella, (eds.) *Software Engineering - ESEC '95. Proceedings of the 5th European Software Engineering Conference, Sitges, Spain*. Springer, Berlin (Lecture Notes in Computer Science 989), pp. 254 - 271, 1995.
- [9] G. Gonenc, "A Method for the Design of Fault-detection Experiments," *IEEE Transactions on Computers*, vol. C-19, pp. 551-558, 1970.
- [10] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, vol. 8, pp. 231-274, 1987.
- [11] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, C. Chen, "Formal Approach to Scenario Analysis," *IEEE Software*, vol. 11, # 2, pp. 33-41, 1994.
- [12] R. Itschner, C. Pommerell, M. Rutishauser, "GLASS: Remote Monitoring of Embedded Systems in Power Engineering," *IEEE Internet Computing*, vol. 2, # 3, 1998.
- [13] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard, *Object Oriented Software Engineering: A Use Case Driven Approach*. Amsterdam: Addison-Wesley, 1992.

- [14] I. Jacobson, "Basic Use Case Modeling," *Report on Object Analysis and Design*, vol. 1, # 2, pp. 15-19, 1994.
- [15] I. Jacobson, "Basic Use Case Modeling (cont.)," *Report on Object Analysis and Design*, vol. 1, # 3, pp. 7-9, 1994.
- [16] W. J. Lee, S.D. Cha, Y.R. Kwon, "Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, # 12, pp. 1115-1130, 1998.
- [17] G. J. Myers, *The Art of Software Testing*. New York: John Wiley & Sons, 1979.
- [18] S. Pimont, J.C. Rault, "A Software Reliability Assessment Based on a Structural Behavioral Analysis of Programs," *Proceedings 2nd International Conference on Software Engineering*, San Francisco, CA, 1976.
- [19] C. Potts, K. Takahashi, A.I. Anton, "Inquiry-based Requirements Analysis," *IEEE Software*, vol. 11, # 2, pp. 21-32, 1994.
- [20] B. Regnell, K. Kimbler, A. Wesslén, "Improving the Use Case Driven Approach to Requirements Engineering," *Proceedings 2nd International Symposium on Requirements Engineering*, York, England, 1995.
- [21] B. Regnell, M. Andersson, J. Bergstrand, "A Hierarchical Use Case Model with Graphical Representation," *IEEE International Symposium and Workshop on Engineering of Computer-Based Systems*, Friedrichshafen, 1996.
- [22] C. Rolland, C. Souveyet, C. Ben Achour, "Guiding Goal Modeling Using Scenarios," *IEEE Transactions on Software Engineering*, vol. 24, # 12, pp. 1055-1071, 1998.
- [23] J. Ryser, S. Berner, M. Glinz, "On the State of the Art in Requirements-based Validation and Test of Software," Universität Zürich, Institut für Informatik, Zürich, *Berichte des Instituts für Informatik* 98.12, Nov 1998.
- [24] J. Ryser, "SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test," to appear as a technical report at University of Zurich, Institut für Informatik, Zürich, 1999.
see: www.ifi.unizh.ch/groups/req/ftp/SCENT/SCENT_Method.pdf
- [25] S. Somé, R. Dssouli, J. Vaucher, "Toward an Automation of Requirements Engineering using Scenarios," *Journal of Computing and Information*, vol. Special issue: ICCI'96, # 8th International Conference of Computing and Information, pp. 1110-1132, 1996.
- [26] I. Spence, C. Meudec, "Generation of Software Tests from Specifications," *SQM'94 Second Conference on Software Quality Management*, Edinburgh, Scotland, UK, 1994.
- [27] A. G. Sutcliffe, N.A.M. Maiden, S. Minocha, D. Manuel, "Supporting Scenario-Based Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, # 12, pp. 1072-1088, 1998.
- [28] K. Weidenhaupt, K. Pohl, M. Jarke, P. Haumer, "Scenarios in System Development: Current Practice," *IEEE Software*, vol. 15, # 2, pp. 34-45, 1998.
- [29] T. Yamaura, "How to Design Practical Test Cases," *IEEE Software*, vol. 15, # 6, pp. 30-36, 1998.

A Practical Approach to Validating and Testing Software Systems Using Scenarios

Johannes RYSER Martin GLINZ
University of Zurich
{ryser, glinz}@ifi.unizh.ch

SCENT: A SCENario -Based Validation and
Test Case Generation Method

5. November 1999

1

Testing as a Problem

- Testing as an unplanned, ad hoc process
 - Sparse time and resources allocated for testing
 - No test plan, tests are not documented
 - Unstructured, non-systematic test case design
- Testing is a drudgerous, error-prone activity
- Lacking tool support
 - Specific or formal testing languages that can only be applied to limited problems or specific domains
 - Automatic testing/test case generation not generally possible
- Lacking method(s), lacking problem understanding

SCENT: A SCENario -Based Validation and
Test Case Generation Method

5. November 1999

2

Solutions to the problem(s) at hand

- Methods and process improvements
 - **Systematic**, methodical test case development
 - Methods that
 - are easy to apply
 - blend into existing development methods/paradigms
 - do not impose inappropriate overhead/intolerable cost
 - Test planning, test preparation and documentation (just do it)
- Improved (formal?) languages and tool support
- **Integration** of testing, **early test case development** and **reuse** of SW artifacts

Our Approach: SCENT

- **SCENT: A method for SCENario-based validation and Test of software**
- Scenario-based
 - User view, black-box view
 - User involvement in requirements engineering
 - Improved communication among stakeholders
 - No special language/notation
 - Requirements capture, documentation and bundling
 - Help the developer acquire the vocabulary of the application domain

Our Approach: SCENT

- Validation
 - Formalization
- Practice-oriented
 - Step-by-step procedure
 - Reuse of (analysis) scenarios
 - No formal language
 - Benefits outweigh additional cost and labor
- Systematic test case development
- Easy integration with software development methods

Scenario Definition

- Scenario – *An ordered set of interactions between partners, usually between a system and a set of actors external to the system. May comprise a concrete sequence of interaction steps (instance scenario) or a set of possible interaction steps (type scenario).*
- Use case – *A sequence of interactions between an actor (or actors) and a system triggered by a specific actor, which produces a result for an actor. A type scenario.*
- Actor – *A role played by a user or an external system interacting with the system to be specified*

The SCENT Approach

- Use of scenarios not only in analysis, but also in testing
 - natural language scenarios to elicit and document requirements
 - ‘formalized’ scenarios (= statecharts) to systematically derive test cases for system test
- ⇒ Reuse of early development artifacts
- ⇒ Integration with existing development methods
- ⇒ Weaknesses of natural language specifications are alleviated by formalization
- ⇒ (Continuous) validation of system

The SCENT Approach

- Statechart/scenario annotations support test case derivation
 - Pre- and postconditions
 - Data ranges and values
 - performance and non-functional requirements/qualities
- Concrete test cases are developed by paths traversal in statecharts
 - ⇒ Enhanced behavioral system specification
 - ⇒ Systematic approach to test case development
 - ⇒ Documentation of test cases

Scenario Creation

- Step-by-step procedure
 - Identify actors, (external) events, in- & output and results
 - Determine system boundaries
 - Create coarse business process level scenarios, prioritize them
 - Refine scenarios by creating a step-by-step description and an overview diagram, and by modeling alternative flows of actions
 - Factor out abstract scenarios, include qualities and performance requirements
 - Have users review and validate scenarios and diagrams
- Scenario Template (layout, structure, format)

Scenario Creation Procedure

| # | Step Description | Results |
|----|--|---|
| 1 | Find all actors interacting with the system | List of actors |
| 2 | Find all (relevant system-external) events | List of events (triggers) |
| 3 | Determine results and output of the system | System output |
| 4 | Determine system boundaries | System boundaries |
| 5 | Create coarse overview scenarios (instance or type scenarios on business process or task level) | List of scenarios |
| 6 | Prioritize scenarios according to their importance and assure that the scenarios cover all system functionality | List of prioritized scenarios Links scenarios – actors |
| 7 | Transform instance to type scenarios. Create a step-by-step description of events and actions for each scenario (task level) | Coarse grained flow of actions in scenarios |
| 8 | Create an overview diagram | Overview Diagram |
| 9 | Have users review and comment on the scenarios and diagrams | Comments and annotations to scenarios |
| 10 | Extend the scenarios by refining the description of the normal flow of actions, break down tasks to single working steps | Description normal flow of actions Hints on test case derivation |
| 11 | Model alternative flows of actions, specify exceptions and how to react on exceptions. Include hints on test case derivation | Alternative flows of actions, exception handling in scenarios |
| 12 | Factor out abstract scenarios | Abstract scenarios |
| 13 | Include performance/ non-functional rqmts./ qualities in scenarios | Scenarios, annotated with qualities |
| 14 | Revise the overview diagram | Revised overview diagram |
| 15 | Have users check and validate the scenarios (Formal reviews) | Validated scenarios |

Example: Scenario Creation

Description: At an ATM the customer may inquire the balance of his/her account or withdraw money up to a certain amount and at given piecing. The customer needs a card and a personal identification number (PIN) to get access to the system and perform the mentioned transactions. The system interacts with a central bank system to get customer and account information and to inquire and update account balances.

Actors: "Customer", "Service Personnel", "Operator", "Banking system"

Events: *Customer inserting card, entering PIN-code, choosing action, entering amount, taking back card, taking cash; operator filling bills; service personnel servicing machine*

System input: *Cards, PINs, choices for actions, amounts, bills*

System output/results: *Cards, balance info, cash/bills*

Scenarios: (1)*Inquire Balance*, (2)*Withdraw cash*, (3)*Service ATM*, (4)*Reload bills*

Excerpts from a Scenario

Scenario 2: **Withdraw cash** (Actor: Customer)

The customer withdraws money

Flow of actions:

1. The customer inserts the card
2. The system checks the card's validity
3. The system displays the "Enter PIN" Dialog
4. The customer enters his PIN
5. The system checks the PIN
6. The system displays ...

Alternatives:

- 1a. The slot is obstructed
 - 1a.1 The customer informs ...
 - 1a.2 ...

Problems in Using Natural Language Scenarios

- How to check for
 - Inconsistencies
 - Incompleteness
 - Ambiguity

⇒ Reviews, Walkthroughs

⇒ Formalization of scenarios

In SCENT we take an intermediate approach:
Formalization of natural language scenarios to a
semiformal graphical representation

Scenario Formalization

- Transformation of narrative scenarios into statecharts
- Transformation is not formal, but supported by heuristics
 - One scenario plus all its alternative flows = one statechart
 - Statecharts are created and refined along with scenarios
 - Model normal flow first, integrate alternatives later on. Check if alternatives are missing
 - Events map to transitions, single working steps in scenarios map to states or transitions in statecharts
 - Check statecharts for internal completeness and consistency
 - Crosscheck statecharts

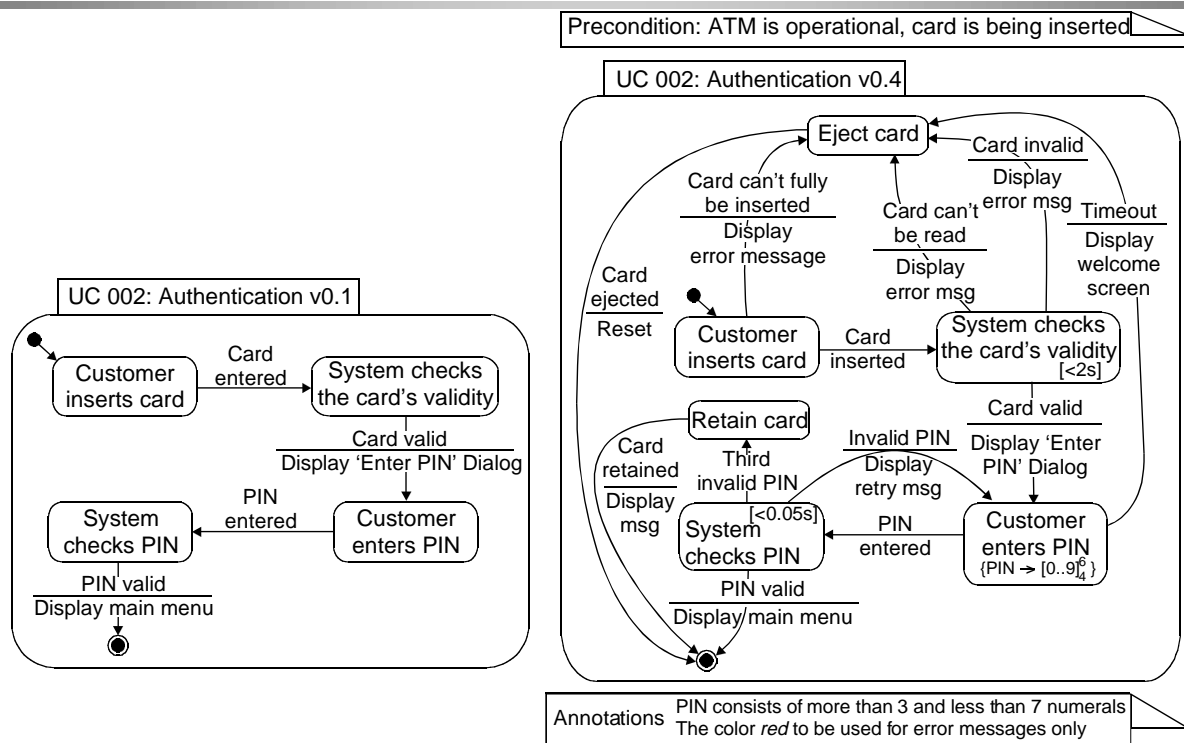
Scenario Formalization

- Formalization makes validation and verification easier
 - Inconsistencies, ambiguities and omissions are found
 - Enhanced validation (animation, ‘formal’ reasoning, ...)
- Statecharts may be integrated to create a model of the whole system
- Validation of statecharts goes hand-in-hand with test case derivation

Statechart Annotation

- Statecharts model system behavior, but data, performance and qualities are not modeled
 - ⇒ Missing information for test development
- ⇒ Annotate statecharts with
 - Pre- and Postconditions
 - Setup for tests derived from statechart
 - Data ranges and values
 - To enable development of domain tests, boundary analysis
 - Expected results (functions as test oracle)
 - Performance and non-functional requirements, qualities

Example: Scenario Formalization



SCENT: A SCENARIO -Based Validation and Test Case Generation Method

5. November 1999

17

Test Case Derivation

- Test case derivation in SCENT consists of:
 - Test case derivation from statecharts
 - Additional testing (supported by scenarios)
 - dependencies between scenarios
 - testing for qualities, performance
 - intuitive tests according to notes in scenarios
- Test cases are determined by path traversal in statecharts
 - node-link-coverage (alternatively switch/n-switch coverage)
 - normal flow, alternative flows, exceptions
- Use of annotations to enhance test case development

Example: Test Case Derivation

1.1 Normal flow is tested first

1.2-1.4 Cover all alternative flows and exceptions

1.3-1.4 Use annotations and notes to develop enhanced test cases

| <i>Test preparation:</i> | | ATM operational, card and PIN (1234) have been issued, card is being inserted | |
|--------------------------|--------------|---|------------------------------|
| <i>ID</i> | <i>State</i> | <i>Input/User actions/ Conditions</i> | <i>Expected output</i> |
| 1.1 | Card sensed | Card can be read, card valid, valid PIN (1234) entered in time | Main menu displayed |
| 1.2 | Card sensed | Card can be read, card valid, invalid PIN (1245) entered in time (first try) | Retry message displayed |
| 1.3 | Retry msg | Invalid PIN (123) entered in time, second try | Retry message displayed |
| 1.4 | Retry msg | Invalid PIN (1234567) entered in time, third try | Card retained, user informed |
| ... | ... | ... | ... |

Conclusions

- SCENT has been applied in two industrial projects
 - Applications to remote monitoring of embedded systems
- Experiences made
 - Use of scenarios proved valuable
 - Iterative refinement appreciated by the developers
 - Scenarios not suited to model system internals
 - Scenario management was a problem
- Direction of future work
 - Tighter integration of non-functional requirements
 - Measure requirements coverage achieved

Structured Testing, a must in a Validation Critical Environment

Nathalie Fuchs

Gitek nv - *interaction through software*

St. Pietersvliet 3, B-2000, Antwerpen, Belgium

Tel: +32 3 231 12 90 - Fax: +32 3 226 10 83

E-mail: nf@gitek.be

1 Introduction

This paper explains our point of view on the subject "Structured testing in validation critical environments" at Janssen Research Foundation.

JRF is a part of Janssen Pharmaceutica, which was founded in 1953. It has 5 products on the WHO list of Essential Drugs. Janssen Pharmaceutica currently produces over 80 pharmaceutical products.

Since 1961 JRF belongs, as a pivot pharmaceutical research company, to Johnson & Johnson. J&J stands for 188 companies in 52 countries with more than 93.000 employees and is active in the consumer, professional and pharmaceutical sector. Of the annual sales, approximately 10% is dedicated to Research and Development. In Belgium the research department is grouped in the Janssen Research Foundation.

The paper covers the following topics: JRF's growing need for a structured test process as part of a business improvement process, the JRF Validation Policy, the incorporation of TMap®¹ in the JRF V-model and the lessons learned of several IT projects and the measures for the future.

2 The growing need

In the pharmaceutical industry, validation became a necessary part of the company's quality system. Validation was defined as follows: "establish a documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes" (FDA², May 1987).

At a time, when "absence of errors" was no longer acceptable as "proof for a validated computer system" the awareness for the need of a structured test process grew quickly.

Therefore, at JRF a V-model was elaborated and the first guidelines for validating computer systems were developed.

By 1997 they were fine-tuned, resulting in a daily used set of guidelines, templates and examples and a JRF validation policy, which is the same as the world-wide approach to quality standards of Janssen Pharmaceutica.

¹ TMap® = Test Management approach®

² FDA = Food and Drug Administration, part of the U.S. government

3 The JRF Validation Policy

At company level the Validation Policy outlines the broad principles for the validation of systems. The implementation of the policy is achieved through a Validation Methodology.

The Validation Methodology encompasses the procedures, methods, tools and techniques that are needed to establish a coherent, pragmatic, and controlled approach across all departments. Together they form a Quality System. The methodology includes an overall description of the validation methodology, the guidelines covering validation during the complete software development life cycle and the common templates. The procedures or standards used, are documented in Validation Master Plans, different for each business area. The Validation Master Plans explain “how” validation is being managed and include an inventory of all types and levels of computerised systems and related processes.

4 The JRF V-model

The JRF Validation policy states that Validation as part of the development process shall assure GXP³ compliance throughout the **full life cycle of a computerised system** (cf. Figure 1: The V-model).

The life cycle includes the following elements:

The Project and Quality Plan (POP)

The purpose of the Project and Quality Plan is to:

- Define JRF’s obligations in relation to quality;
- Determine all the means that are and will be applied to meet the JRF’s technical and quality requirements;
- State all the participants of the project, procedures, rules and applicable methods.

The Validation Plan (VP)

The Validation Plan describes the process to be validated, and how validation will be conducted. The completion of the plan will produce the Validation Report, which summarises the results of the validation activities and defines the final validation status.

The objective of the Validation Plan is to provide a basis for planning and control of the activities required, to validate the concerned system.

The User Requirements (URS)

The User Requirements Specification defines clearly and precisely:

- What the process in which the system is to be used requires from the system;
- The business and user needs for the system;
- The constraints that are set for the execution of the project;
- The company standards (both tools and methods) that have to be followed when developing and implementing the system.

The URS is an integral part of the Software Development Life Cycle (SDLC). This means that it will be continuously updated during the lifetime of the computerised system to reflect the true functionality as built and installed.

³ GxP regulations: Good Manufacturing Practices (GMP), Good Laboratory Practices (GLP), Good Clinical Practices (GCP). The paper stresses experience in a GLP compliant environment.

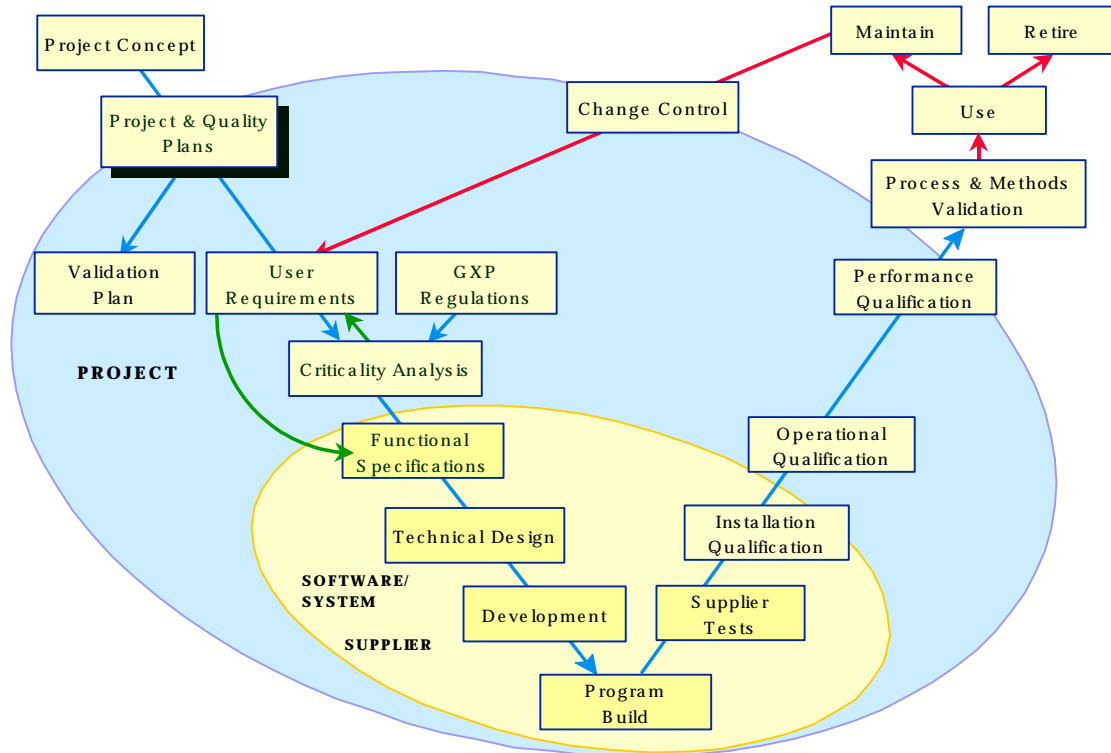


Figure 1: the JRF V-model

The Criticality Analysis (CA)

The purpose of the Criticality Analysis is to:

- Highlight the items which are critical in regard with the GxP regulations and identify the GxP regulations that are applicable to the concerned system;
- Show that the user requirements of the concerned system take into consideration the identified regulations and thus demonstrate that the concerned system is GxP compliant;

Define the subsequent actions that must ensure that critical areas are appropriately addressed. Areas that are not well covered in the current version of the requirements have to be included in a subsequent version of the URS-document.

The Functional Specifications (FS)

The Functional Specification identifies how the system will meet the User Requirements Specification. Traceability between URS and FS has to be maintained and this is achieved through a traceability matrix and numbering of all specifications. The FS will define the system’s functions, including explanation of how they will be performed manually (in accordance with procedures), by equipment or by a computer system or systems.

The Technical Design (TD)

The Technical Design identifies in precise technical detail how the system will meet the Functional Specification. Traceability between URS, the FS and the Technical Design has to be maintained and this is achieved through a traceability matrix and numbering of all specifications.

Supplier Tests (ST)

The Supplier tests incorporate code reviews, unit, integration and system testing by the supplier in a development environment.

Installation Qualification (IQ)

The objective of the Installation Qualification is to verify that the system has been installed and configured in its operational environment according to the design documentation and recommendations of the manufacturer or supplier(s).

Operational Qualification and Performance Qualification (OPO)

The objective of the Operational Qualification is to demonstrate, through the execution of tests, that the system performs and behaves as specified. The objective of the Performance Qualification is to prove that the entire system performs correctly and consistently in the true operating environment, according to the requirements for the process.

This is managed using an OQ and PQ Test Plan. The plan must:

- Be written and approved prior to execution;
- Ensure that all activities are carried out and documentation has been delivered;
- Summarise all “out of acceptance” results (known errors) in a document “Test Report”;
- Be followed by a Validation Report.

Validation Report (VR)

The Validation Report summarises the results of the validation activities and defines the final validation status.

The objective of the Validation report is to confirm that the items required by the Validation Plan have been completed. The Validation report makes the overall conclusions for the project.

The Validation report must:

- Be written and approved;
- Enclose all out of acceptance results (known errors) with a status “Resolved”, “Part of a Corrective
- Describe all non-compliance remarks in relation to the departmental SOP’s⁴.

The system may not be released as fully validated if the Validation Report has not been approved by Quality Assurance.

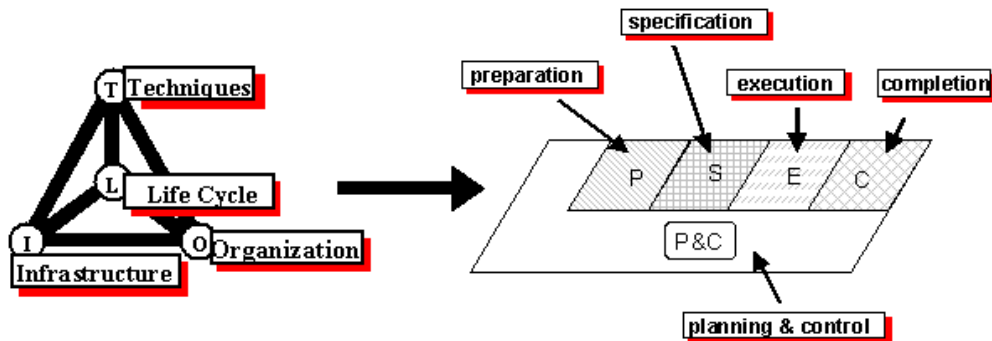
⁴ SOP = Standard Operating Procedure

5 The introduction of Test Management approach®

JRF refined the test aspects of the V-model by incorporating TMap® in their validation methodology. TMap® was developed by IQUIP Informatica B.V. in the Netherlands.

TMap® is a generic model based on 4 cornerstones:

- A development process related life cycle model for the testing activities (L). The model recognises five phases: planning and control, preparation, specification, execution and completion;
- Solid organisational embedding (O);
- The right resources and infrastructure (I);
- Usable techniques for the various testing activities (T).



For several pharmaceutical projects the life cycle model was incorporated in the Operational and Performance Qualification of the JRF V-model.

6 The lessons learned and measures taken

6.1 The Test Life Cycle Model

6.1.1 Planning & Control

During the very first project, several mismatches between TMap® and the JRF Validation Methodology appeared. TMap® calls for a very early development of a high-level test plan while JRF defined a more detailed test plan later in the project.

There were no detail plans drawn up. This was met by planning and re-planning the testing activities on a regular basis. In the near future detail plans will be made. For every part of the detail plan the used test specification techniques, the kind of test and the applied test base will be mentioned.

Moreover, the plans will indicate the planned effort and the predicted run time. Optionally the predicted number of defects can be included (cf. “Experience based estimations”, TESPRA Johan Swinnen Gitek nv).

Planning and control was well organised from the testing point of view although the first plans were too optimistic. Planning and control was formalised on a weekly basis and communicated to all members of the test team. Today an extension of the working hours administration is taken into consideration since the testing hours alone can not give a detailed cost status of the testing project (cf. “Experience based estimations”, TESPRA Johan Swinnen Gitek nv).

Both methodologies proved to be rather flexible, some high-level test information can be incorporated in the Project and Quality Plan and the test plan will be created earlier during validation.

6.1.2 Preparation

During our first project, the preparation phase was impeded because of problems by the version control of the test base and the objects to be tested. Meanwhile awareness is created and the need for good version control is recognised.

From the testing point of view version control was well managed.

JRF is now thinking of elaborating a tool to overcome the former administrative tasks about version control. (The disadvantage of the use of a tool is that it can have a rather restrictive impact).

JRF uses version control whereby documents are provided with a document name, version number, release number and fix. The fix can alter during the entire project where as the release number only alters in case of partial release in production. The version number alters in case of complete changes.

6.1.3 Specification

During specification the test cases were specified and the appropriate infrastructure was put in place. The test cases were defined in two stages, the logical and physical test design. The used test specification techniques were conclusive. JRF started for the second pilot project to execute detailed pre-tests. The supplied products were tested during the specification phase to determine whether it would serve any purpose to submit the test object to the formal structured test. This resulted in a shorter and more effective execution. Fewer defects were found and less re-tests had to be performed.

Experience indicates that a more formal test environment is needed. One of the aspects needed in such an environment is a tool restricting the access to the test base.

6.1.4 Execution

The defect tracking proved that the phase of test execution was productive and effective. There were a lot of defects made on test execution. This was due to incorrect and incomplete answers on the intakes and no solution for a considerable number of defects with a low severity rather than the design of the scripts. New versions of test scripts had to be made for re-test. This resulted in a loss of efficiency.

The JRF Validation Methodology stresses the importance of functional and technical documentation. Good communication and well-thought answers concerning design and development is very important. The changes seen during the development of the sequential projects proves that this is very well possible and can be changed in a short time period.

According to the JRF Validation Methodology a test witness is compulsory during test execution if test evidence can not be gathered in other ways. The related need of extra manpower, causing extra costs, was solved using Lotus ScreenCam®. This tool is used to create a kind of video movie that registers the actions executed on a PC. The resulting file is then stored on a CD-ROM. JRF is implementing the use of this tool in the entire organisation.

The type of project did not allow a separate test environment, since this would require the purchase of a full set of lab-instruments. Analysing samples and other research work took priority.

6.1.5 Completion

Planned completion time was insufficient when re-tests were taken into consideration. Too many findings were made on planned but not executed modifications. This underestimation of the number of defects was the consequence of too little attention from developer's point of view and too much confidence from the side of the test team. As a result JRF's Methodology has been adapted and now states that white-box testing should be done during development before software is released for black-box testing. A planning with checkpoints and milestones is elaborated for Development.

6.2 Techniques

JRF is aware that the intake (inspection) of the test base has to be executed earlier in time. Fagan inspections are considered.

The syntactic tests were drawn up with too much detail. In the future they will be put together in one script where possible.

Depending on the risks and use of the test base, the dataflow and error guessing part of the test scripts must be extended with an elementary comparison test.

Detachment of the logical and physical test specifications has proven its advantage. Certainly in case of regression tests.

6.3 Infrastructure

Clearly the test infrastructure is a point of attention. Not only a separate test environment is needed to test efficiently, the PC's used by the testers are of importance as well and sufficient resources have to be spent in this area.

JRF decided not to use test tools yet since they are convinced that a test process needs to be in place before automation is worthwhile.

The defect administration was very effective and userfriendly. JRF is introducing it through the complete organisation. In the future defects will be managed using a database instead of an excel spreadsheet. This would allow minimisation of manual tasks and the improved use of metrics over several projects (cf. TESPRA, Johan Swinnen, Gitek nv).

The use of metrics is introduced. Questions like “where to find the causes of the defects” and “which actions should be taken to structurally solve problems” are answered. JRF recognises the great importance of an estimation of the expected defects to budget a project. The defects have to be linked to change management.

6.4 Organisation

The JRF organisation has clearly found a good test methodology by incorporating the TMap® approach in several projects. JRF has decided to choose TMap® to test all major software releases.

The integration of a test methodology in the JRF Validation Methodology clearly needed time and effort. At this moment, planning and formalisation of decisions made, is included in standard project management.

Prototyping at development level has proven its advantages. JRF stressed the importance of white-box testing, even when prototyping.

The test team was well organised. Co-operation and communication were open and clear although the instability of the test base caused high work pressure at moments.

7 General objectives for the future

For the near future JRF is concretising actions on the following issues:

- Elaborating a Janssen Pharmaceutica generic V-model with the clear incorporation of the TMap® approach;
- Continue to cultivate the awareness that a validation process is an integrated part of the development process. More stable software should result in less maintenance. Therefore clear roles and responsibilities for all project members must be set;
- Implementing validation and starting test planning at a higher stage of the V-model (cf. The future V-model). This must result in reducing costs, more predictable and controllable budgets or projects;

- Elaborating the requirements by levelling the importance (high, medium, low). This in order to specify the business risk in detail;
- High quality test evidence and important test metrics by the use of defect tracking (cf. TESPRA Johan Swinnen Gitek nv) and other tools to provide a clear view of the overall test and development process. This, in order to take the right management decisions at the right time;
- Elaborating traceability matrices between user requirements, functional specifications and logical test cases. This in order to clearly explain “why” certain functions are in place and to prove that the information system is under control.

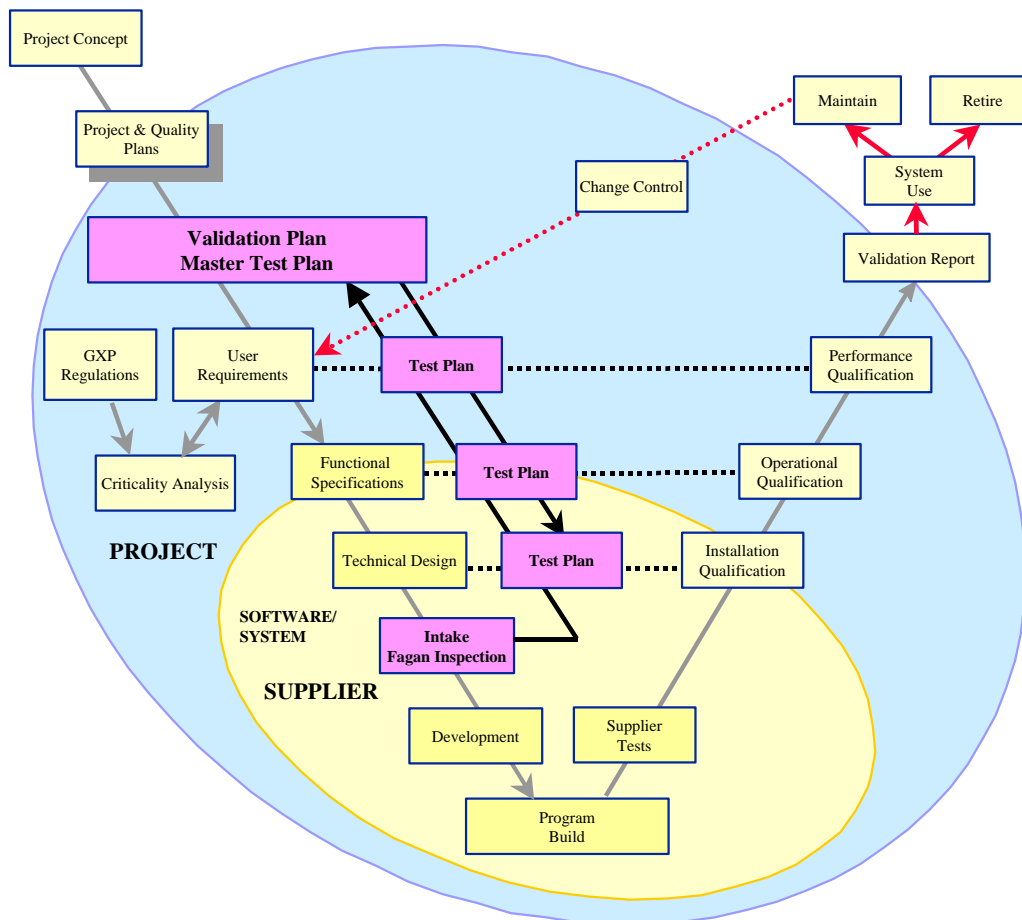


Figure 2: the future JRF V-model

8 Biography

Nathalie Fuchs is a commercial engineer and has a post university degree on Information Management. In the early nineties she started as a project leader – application manager practising testing information systems in the retail business. She has experience in testing and implementing software for cash register systems, change management and user support. She joined Gitek nv in the beginning of 1998 and is now test co-ordinator of several pharmaceutical test projects.

9 Appendices

9.1 Project roles and responsibilities

| | |
|---------------------------|--|
| System sponsor | <ul style="list-style-type: none">• Ensures that regulatory requirements are met;• Provides adequate resources;• Reviews the qualification status of the system prior its release for operational use. |
| System User (Key User) | <ul style="list-style-type: none">• Executes the qualification test scripts;• Uses the system according to the SOP's. |
| System Owner | <ul style="list-style-type: none">• Ensures that regulatory requirements are met;• Ensures user training;• Ensures the availability of all documentation;• Corrects definition of User Requirements• Evaluates recommended change actions;• Ensures the availability of the SOP's and guidelines for system use;• Ensures correctness of data conversion;• Ensures controlled access to the system;• Qualifies of the system prior to operational use;• Maintains the qualification status of the system. |
| System Validation team | <ul style="list-style-type: none">• Prepares the URS, VP, IQ, OPQ, test scripts and Validation Report;• Supports the execution of the test scripts;• Resolves issues arising from the validation tasks;• Performs project verification. |
| Quality Assurance | <ul style="list-style-type: none">• Audits validation activities for compliance with the validation policy;• Reviews the URS, VP, IQ, OPQ, test scripts and Validation Report. |
| System Developer | <ul style="list-style-type: none">• Prepares the System Specifications with the owner;• Prepares the System Design Description and the system documentation;• Provides documented evidence of the product quality during development and installation;• Provides technical support. |
| System Supplier | <ul style="list-style-type: none">• Provides documented evidence of the product quality during development and installation;• Provides technical support. |
| LSV Manager | <ul style="list-style-type: none">• Provides validation guidelines;• Provides project and validation standards and templates• Assists during validation. |

9.2 Glossary

| | |
|----------------------------|--|
| Intake | Formal review / audit of all specifications (test base) for the system that is to be developed in which consideration is given to the question whether the documentation is sufficiently complete accurate and consistent to serve as the starting point for the tests. |
| Fagan Inspection | Formal review / inspection of the test base executed by several users (e.g. end user, developer, test team, ...). |
| Physical test case | A physical description of the test inputs and database at the level at which the test cases can be executed. The description also includes the test actions to be performed and the way in which the expected and actual results may be compared. |
| Syntactic test | The object of a syntactic test is to detect defects in the layout of screens and reports and prints, and in primary input checks relating to the entry fields. Standards in force and specific descriptions of screens and reports in the functional specifications may be used as test criteria. This test technique is particularly useful in testing on-line systems, but may also be used for checking reports produced in batch-oriented systems. |
| Dataflow test | As the name suggests, data flows and their processing form the basis of a data flow test. It is a test technique whereby processing by functions and relations between functions are tested. A data flow test is a semiformal test technique used in testing functionality as part of a black-box test, especially during acceptance testing. |
| Error guessing | Error guessing is basically unstructured testing and consists of guessing at the occurrence of defects. Its value lies in the unexpected: it includes tests, which in all probability would not have been executed otherwise. It makes a valuable contribution to the structured test specification techniques. The error guessing technique may be used in all test levels and may basically focus on any kind of quality characteristic. It is not advisable to use error guessing for a relatively unstable test object since, as a result of the very nature of the technique, repeatability, and therefore the reproducibility of defects, is very low. |
| Elementary comparison test | In an elementary comparison test, processing is tested in detail. The test verifies all functional paths of a function. An elementary comparison test guarantees a high degree of completeness, and is therefore time consuming. It is used mainly for testing functions considered to be of major importance, and for complex calculations. This test technique was developed basically for black-box testing, but it may also be used in a white-box test. Elementary comparison tests focus on the quality characteristic functionality. |
| Regression test | Selective testing to verify that modifications have not caused unintended adverse side effects or to verify that a modified system still meets requirements. Regression testing aims at establishing confidence that the quality of the system is not deteriorating. |



“Structured Testing, A Must in a Validation Critical Environment”

Nathalie Fuchs
Gitek nv



Agenda

- Janssen Research Foundation
- The JRF V-model
- The incorporation of TMap®
- Lessons Learned
- The future



Janssen Research Foundation

- Foundation at 1953
- 5 products on WHO of essential drugs
- Belongs to Johnson & Johnson since 1961:
 - 188 companies in 52 countries
 - with 93.100 employees
 - active in 3 sectors: consumer, professional & pharmaceutical



JRF: Definition of Validation

- FDA Definition (1987)

Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes

- Key validation concepts
 - Process-based
 - Establish a plan
 - Pre-determined specifications
 - Document the evidence
 - Consistent performance



Ref. FDA 'Guideline on General Principles of Process Validation (May 1987)'



JRF: Validation Policy

- Worldwide CS Policy - Jan 1997 and Dec 1998
 - Applicable to all Janssen-Cilag and JRF

Computerised systems have become critical to our business, and it is important that their quality meets our needs.

The purpose of the Policy is to ensure that we can

- trust the systems that can affect product quality, and prove reliability
 - trust our data and data handling systems, and prove that they are reliable
-
- CS Validation Policy is supported by:
 - Methodology, Guidelines and Templates
 - Often adapted to local situations and implemented at Company / Department level



Structured Testing, a Must in a Validation Critical Environment, nr. 6

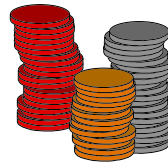
innovation through software

– Best practice examples



JRF: Why do validation?

- Increases reliability of system
- Increases confidence of the user
- Increases the quality and decreases downtime
- Required by the authorities
- Improves the image and the reputation of the company
- Reduces the time to complete a Project properly
- Return On Investment
 - Cost of validation is not easy to measure
 - What is the cost of fixing errors ?
 - What is the price of bad quality ?
 - Can you take the risk of not validating ?

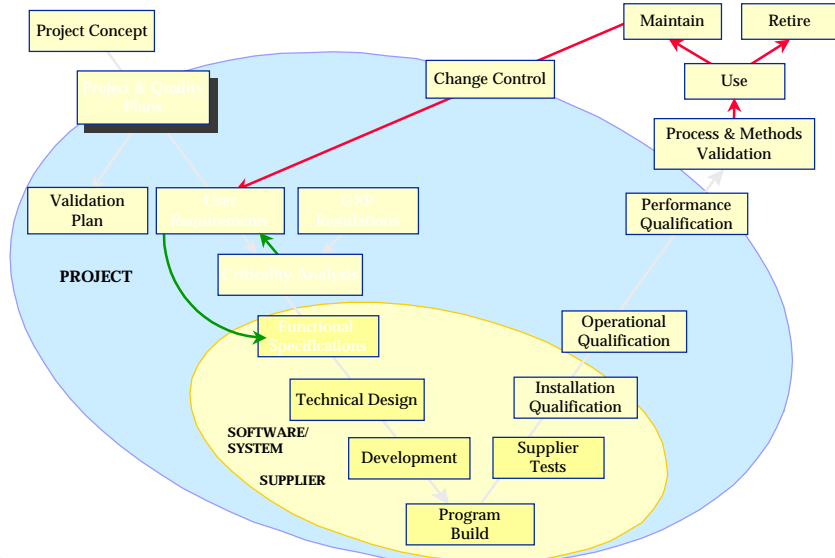


Structured Testing, a Must in a Validation Critical Environment, nr. 6

innovation through software



JRF: The V-model



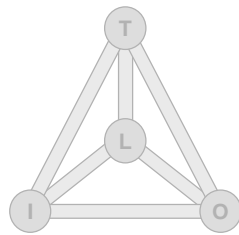
gitek®

Structured Testing, a Must in a Validation Critical Environment, nr. 7

interaction through software



JRF: The incorporation of TMap®



| | |
|---------------|----------------|
| What, when ? | Life-cycle |
| How ? | Techniques |
| Where, etc. ? | Infrastructure |
| Who ? | Organisation |

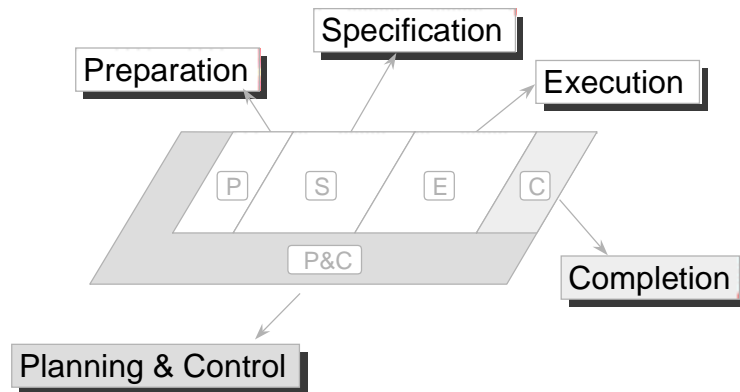
gitek®

Structured Testing, a Must in a Validation Critical Environment, nr. 8

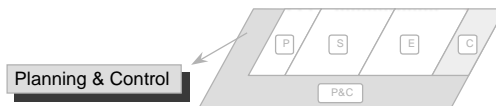
interaction through software



JRF - TMap®, the Life Cycle



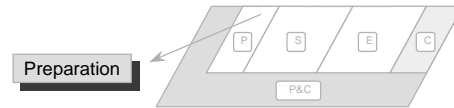
JRF - TMap®, the Life Cycle



- First project mismatch TMap® and JRF Validation:
 - detailed test plans later in the project as TMap® indicates
 - both flexible
 - high-level test information in Project & Quality Plan
- No detailed test plans:
 - solved by regular re-planning
 - new: planned effort and predicted run time in detail plans
 - new: predicted number of defects in detail plans
(cf. TESPRA, Johan Swinnen, Gitek nv)
- Planning and control well organised from testing point of view
 - too optimistic in the beginning
 - extension of total workload administration



JRF: Lessons Learned



- First project: preparation phase was impeded because of:
 - problems with version control of test base and test objects
 - meanwhile:
 - awareness created
 - good version control recognised
 - testing point of view: version control well managed
- JRF is now:
 - elaborating a tool to overcome former administrative tasks about version control (disadvantage: a rather restrictive impact)
 - using version control whereby documents with name, version number, release number and fix



JRF: Lessons Learned



- Test cases were specified in two stages: logical & physical
- Used test specification techniques conclusive
- Appropriate infrastructure put in place
- Second pilot: execution detailed pre-tests
- More formal test environment needed with tool restricting the access to test base



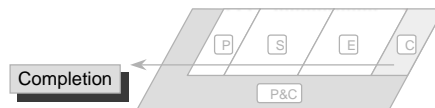
JRF: Lessons Learned



- Defect tracking proved productive and effective test execution
- Lot of defects on test execution due to:
 - incorrect and incomplete answers on the intakes
 - no solution for a considerable number of defects with low severity
 - new versions of scripts for re-test: loss of efficiency
- The JRF Validation Methodology stresses:
 - importance of functional and technical documentation
 - good communication of well-thought answers concerning design and development
- Compulsory test witness solved using Lotus ScreenCam®
- Type of project did not allow separate test environment



JRF: Lessons Learned

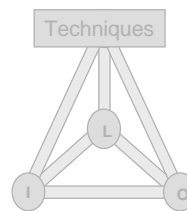


- Planned completion time insufficient when re-tests into consideration
 - Too many findings on planned but not executed modifications due to:
 - too little attention from developer's point of view
 - too much confidence from the side of the test team
- as a result:
- JRF's Methodology adapted and states that white-box before release for black-box testing
 - planning with checkpoints and milestones elaborated for Development



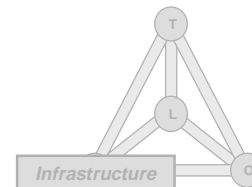
JRF: Lessons Learned

- JRF aware: intake of test base earlier in time
- Fagan inspections considered
- Syntactic tests with too much detail
- Future: together in one script where possible
- Detachment of logical and physical test specifications
⇒ regression tests



JRF: Lessons Learned

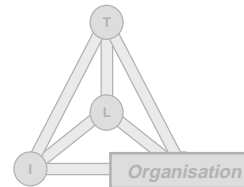
- Infrastructure point of attention:
 - tester's PC: sufficient resources
 - separate test environment needed to test efficiently
- JRF: no use of test tools yet:
 - test process in place before automation is worthwhile
- Defect administration:
 - JRF introducing in complete organisation
 - Future: use of metrics* over several projects
 - Questions like “where to find causes of defects” and “which actions should be taken to structurally solve problems” are answered
 - Expected defects to budget a project*
 - Linked to change management



*cf. TESPRA, Johan Swinnen, Gitek nv



JRF: Lessons Learned



- JRF clearly found a good test methodology by incorporating TMap®
- JRF decided to choose TMap® to test all major software releases
- Integration of a test methodology in the JRF Validation Methodology clearly needed time and effort
- At this moment, planning and formalisation included in standard project management
- Prototyping at development advantages nevertheless JRF recognises the importance of white-box testing
- Test team was well organised. Co-operation and communication open and clear although the instability of the test base caused high work pressure

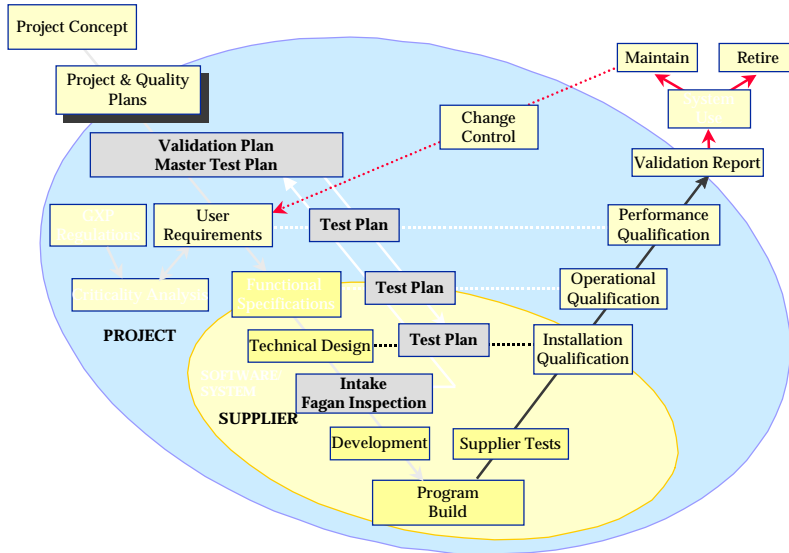


JRF: The future

- Generic V-model with clear incorporation of TMap®
- Continuing cultivating awareness: validation process is integrated part of development process
- The future V-model: test planning at higher stage :
⇒ reducing costs, more predictable, controllable budgets
- “Peer reviews” of URS, FS, TD before final budget
- In order to specify business risks in detail, levelling importance of the URS(high, medium, low)
- High quality test evidence and important test metrics for right management decisions (cf. TESPRA Johan Swinnen Gitek nv)
- Traceability matrices URS, FS and logical test cases:
 - to clearly explain “why” certain functions are in place
 - to prove that information system is under control



JRF: The future V-model



???

“Structured Testing, A Must in a Validation Critical Environment”



Nathalie FUCHS
 Gitek nv
 Sint-Pietersvliet 3
 B - 2000 Antwerp
 Belgium
 nf@gitek.be

Improving developer's tests

by Tim Koomen and Rob Kuijt

IQUIP Informatica B.V.
PO Box 263, 1110 AG Diemen,
The Netherlands

phone: +31 20 660 66 00
fax: +31 20 695 32 98

email: koomenti@iquip.nl / kuijtrob@iquip.nl

Abstract

This paper discusses the reasons for improving the developers' tests. Next, a practical approach is given for improving the quality of the developers' tests (i.e. program test and integration test). The emphasis of the approach is on:

- clear testing responsibilities;
- the idea that improvements come from within the development team;
- insight into the quality of the test object by using exit- and entry-criteria.

The paper concludes with two case-stories.

Introduction

Testing theory states the importance of early testing. One of the earliest forms of testing is performed by the developers. There is sufficient literature on how this kind of testing should be performed, including a lot of techniques and tools. However, there is a large gap between theory and practice. In practice, developers often test based on their intuition and in an unstructured way, in stead of using techniques, etc.

Based on our own experiences, this paper gives reasons why better developer testing is important and how improvements can be implemented. The paper concludes with two case-stories.

Our experiences originate from the world of administrative automation (financial institutions, government, industry), and "new media" projects (such as internet, knowledge management, datawarehousing).

Characteristics of developers' tests

Typical developers' tests are the program test and the integration test. Some characteristics of these tests (and differences with other types of tests) are:

- The finder of a defect is also (often) the solver of the defect. This means communication overhead can be kept to a minimum;
- (In principle) all defects found should be repaired before the product is handed over. Therefore, the amount of reporting is limited;
- A developer differs in attitude from a professional tester; the former wants to demonstrate that the product works, the latter wants to demonstrate the inverse. Time-consuming and thorough testing conflicts with the developer's attitude;
- The tests are an integral part of the development process;
- At the time these tests start, most defects are in the product: this requires cheap and fast repair of defects.

Should developers' tests be improved?

When asking the developers whether their ways of testing need improvement, frequent answers are:

- Not enough time, too expensive;
Only if given more time and money, better testing is possible. However, development is always short of time and money. But when the project leader is asked for more time and money, that person responds something like: "If I give the developers more time and money, they'll certainly spend it. The big question is, on what?")
- Faith in current way of working, in current product quality;
It is common knowledge that 100% defect free software is impossible to achieve. Apart from that, developers are proud of the products they develop.
- A good (better) test follows
Later tests are performed by test professionals. These people are trained in testing and they typically find a lot of defects. And what's more important, they like testing. Because for most developers ...
- Testing is boring!

There are of course a number of reasons for improving developer testing:

- Defects one finds oneself are easier to analyse than defects found by other parties;
- Early rework is cheaper, since knowledge is still fresh and all relevant parties are still there;
- Earlier feedback prevents similar errors;
- White-box techniques used in developers' tests find different defects from black-box techniques used in later tests.
- Because higher quality products are delivered, less defects are found in later tests (and in production). As a consequence, developers can spend less time reworking products and more time creating products;
- One of the most uncertain planning factors is the amount of time and resources necessary for rework activities. More certainty about product quality therefore results in better project planning;
- For the same reason, the project lead time is shortened;

These arguments sound good enough, but in practice seldom win from the arguments given against improvement. However, there is a new development adding arguments in favour of improving. This important development is the increasing strategic use of software, i.e. supporting the primary business of an organisation, directly dealing with the customers and readily adaptable to new challenges. This means a different kind of customer than the traditional IT department, demanding quality software delivered in time. Improving only system and acceptance testing will result in failing to meet these "in time" demands. Improving developers' tests is an important step towards meeting the demands.

How to improve? An approach

Below, a practical approach on how to improve developers' tests is described. The approach has the following key aspects:

- organisation
- use of test design techniques
- use of a test life-cycle model

1) Organisation

Perhaps the most important aspect of the approach is that an **Application Integrator (AI)** will be made responsible for the progress of **integration** and for the **quality** of the outgoing product. The AI negotiates with the project manager or the development team leader what level of quality is required: under what conditions can the system be released to the next phase (exit-criteria). The AI also demands insight in the quality of incoming modules or programs (entry-criteria). A module or program is only accepted into the integration process if it meets the entry-criteria. In order to prevent mixing interests, the AI should not be development team leader. This creates a deliberate tension between the AI, who is responsible for quality, and the development team leader, who tends to focus on functionality and the amount of time and resources spent. Since the AI is a member of the development team, this generates far less resistance from the developers against improving their testing than other test approaches would do and considerably raises the awareness of quality within the development team. For test specific expertise, the AI is supported by test professionals (outside of the project). The AI communicates with the customer on quality issues. Because communication is essential, it is very important for the AI to have good social skills.

2) Use of test design techniques

The approach does not prescribe 100% use of formal test design techniques as this will (in our experience) generate too much resistance. Instead, the AI, program testers and project manager or development team leader negotiate: important parts of the system will be tested using formal test design techniques, less important parts will be tested using informal techniques or even in the old-fashioned, undocumented way of testing without use of any techniques. A good balance has to be found, for which several aspects play a role:

- risks for the organisation / importance of the system;
- desired quality of the product;
- progress of the project;
- test maturity of the development team;
- test coverage;
- test evidence;
- resource consumption (of using the test design technique).

Popular techniques often applied in our projects are checklists and marking test situations in the functional specifications. Only test situations too complex to be tested directly from these markings are detailed further into specific test cases. Although definitely not watertight, the marked up documents supplied with the tester's initials serve as test evidence.

3) Use of a test life-cycle model

Exit- and entry-criteria and other agreements are laid down in a test plan (phase 1), test cases are prepared (phase 2) and executed (phase 3). Again, practical use prevails: writing documentation such as a test plan is not a target in itself, but serves as a means of communication between project leader, AI and customer, and should be kept as minimal as possible.

Case-stories

1. Telebanking

A large bank ordered a Telebanking application (size: 900 function points) to be developed. After some time of developing, the product is handed over to the acceptance testers of the bank. Acceptance testing starts and after 3 man weeks of testing, they have registered 341 defects in the product. The developer is

dismissed... Next, IQUIP gets the assignment (starting from scratch). To avoid the first experience from happening again, delivering a quality product in time is considered crucial. This is the first time the function of AI is used in a project. The team then consists of 7 people (development team leader, AI, 5 programmers). This time, the acceptance testers have a more difficult job. After 12 man weeks of testing, 130 defects have been registered. These results are even more promising when we look at the distribution of defects in severity categories:

| Severity | Very serious | =====> | | Cosmetic |
|------------|--------------|--------|-----|----------|
| Competitor | 10% | 15% | 30% | 45% |
| IQUIP | 0% | 10% | 50% | 40% |

2. Mail room

The second case involves a very small project for Dutch mail, with only three developers and a project manager. The purpose of the project was to develop two small applications for mail room control (sizes: 410 and 274 function points). With such a small team, a full-time AI was not feasible. The project manager solved this by attributing the AI role to the two senior developers (one for each application) and implementing the other improvements as practically as possible. (If only one application was to be developed, one developer would get the role of AI, but his or her programming results would be reviewed by the other developer.) This approach turned out to be a success. During acceptance testing 33 potential defects were registered, 9 of which turned out to be no defect at all. During production, so far only 1 defect has been found. The big advantage to both customer and project manager is the consistent quality of the applications, which has become less dependent on the skills of the individual programmer.

Advantages

Of course, the used numbers such as in the case descriptions above raises all kinds of questions (in case 1 maybe the competitor was really doing a bad job, 130 defects still sounds like a lot, etc.), but the main point is: developers, project leader, customer and application integrator were enthusiastic about the approach and felt that it worked for them. Developers became more aware of the importance of quality (and how to achieve this), the project manager had better control over the project, the customer had more confidence in the quality of the application, and as for the AI: the job was fun and rewarding, requiring communicating with a lot of different people.

Conclusion

This paper describes the importance of better testing by developers, and shows a practical way to achieve this. In practice the approach turns out to be successful since:

- clear testing responsibilities are defined;
- the improvements come from within the development team;
- there is insight in the quality of the application under test by using exit- and entry-criteria.

Improving developer's tests

Tim Koomen and Rob Kuijt
IQUIP Informatica BV
The Netherlands
koomenti@iquip.nl / kuijtrob@iquip.nl

Agenda

- What are developer's tests
- Why improve?
- How to improve?

Agenda

➤ What are developer's tests

- Why improve?
- How to improve?

Characteristics

- Low-level tests: unit test, integration test, mostly performed by developers
- Defect finder = defect solver
- All defects should be repaired (no risk reporting)
- Attitude developer versus attitude tester
- Tests integral part of development process
- Most defects in product: fast & cheap repair

Agenda

- What are developer's tests
- ➔ Why improve?
- How to improve?

No improvement necessary!

- Good enough as it is
- Not enough time, too expensive
- A good (better) test follows!

- **“Not my job”**

Benefits

Benefits to developer:

- Easier and cheaper repair
- Prevention of (similar) defects
- Less later rework
- Better planning
- Shortened lead time
- More certainty about product quality
- Better product quality

Benefits to total project:

$$1 + 1 = 3$$

Improvement is necessary

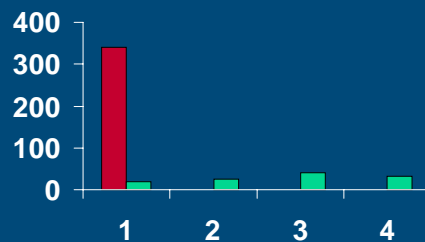
Customer demands
QUALITY software
IN TIME!

Case

- Large bank
- Telebanking application: 900 function points
- Second try
- Clean start
- Team: 7 developers
- Duration: 8 months

Case results

- Defects from acceptance testing:
 - First try: **341 defects in 3 manweeks**
 - Improved testing: **130 defects in 12 manweeks**

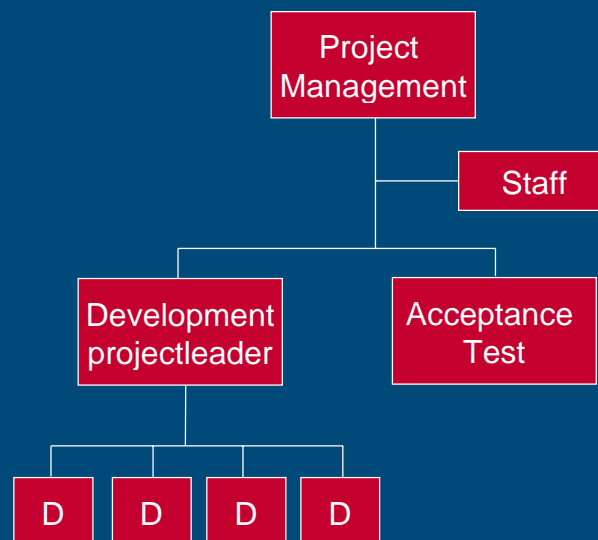


- Severity category: serious \longrightarrow cosmetic
 - First try: 10% 15% 30% 45%
 - Improved testing : 0% 10% 50% 40%

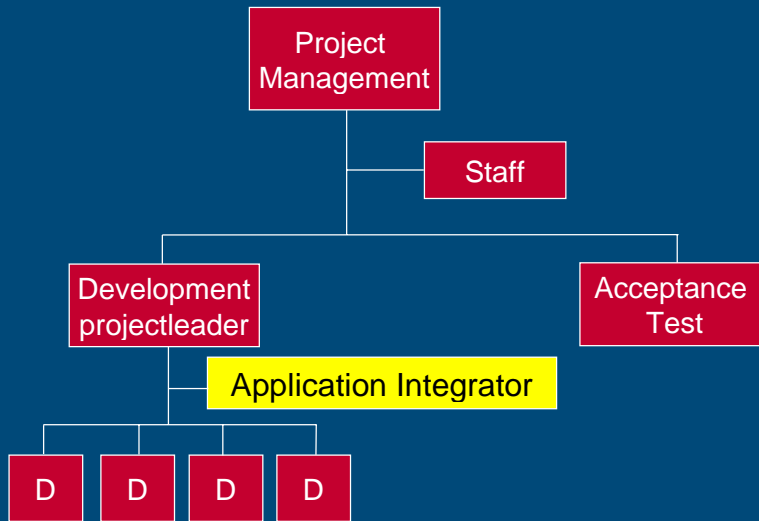
Agenda

- What are developer's tests
- Why improve?
- ➔ How to improve?

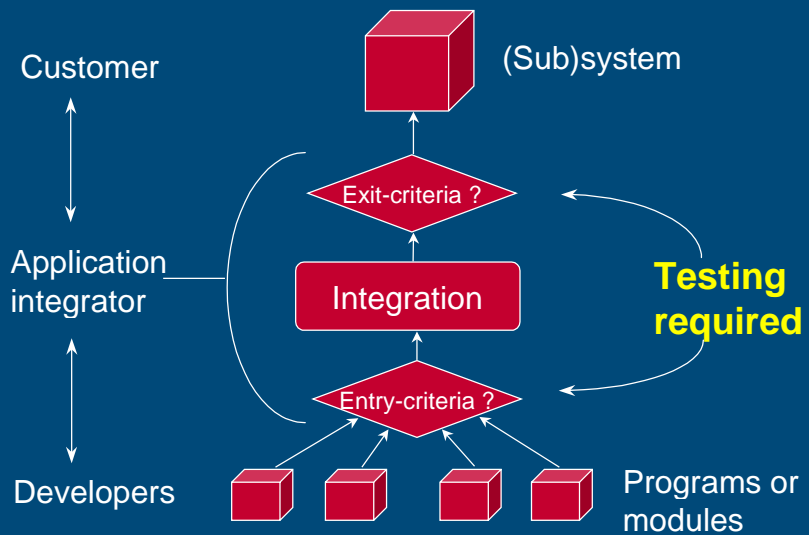
Traditional organisation



Role: Application integrator



Quality and integration



Test strategy

- Be practical!
- “Right quality”
- Important functions:
 - better test coverage
 - more test evidence
- Important quality characteristics: the same
- Popular “techniques”
 - Checklists
 - Marked up functional specifications
 - Only specification of test cases if necessary

Use of a life-cycle model

- Test planning phase
 - agreements
 - strategy
 - ...
- Test specification phase
 - identify test situations, according to strategy
 - (if necessary) prepare test cases
 - ...
- Test execution phase
 - execute tests
 - analysis
 - reporting
 - conserve required evidence
 - ...

Summary

- Improving developers' tests is necessary:
 - Better product quality
 - Insight
 - Customer satisfaction & confidence
 - Quality awareness developers
- How to improve:
 - Application Integrator
 - Practical
 - Improvement from within development team, supported by professional testers



Do you really think...

Can you explain...?

Questions?

Who!

Ask now!

or email:

How?

koomenti@iquip.nl

kuijtrob@iquip.nl

Where...

What??

Developing Embedded Software for a Helicopter Testbed

Henrik Oertel, Klaus Alvermann, Stephan Graeber, Lothar Thiel
Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)
Institute of Flight Mechanics
Braunschweig, Germany

Abstract

The Institute of Flight Mechanics of the German Aerospace Center (DLR) develops software intensive real-time systems for embedded airborne systems. Based on the current development of an experimental on-board equipment for a new helicopter testbed, the Process Improvement Experiment 27751 ICARUS enables the usage of the design tool ObjecTime. This paper gives an overview over the helicopter baseline project, presents the objectives as well as the approach of the experiment, describes the software development procedure and explains the evaluation of several design tools, analyzes the key lessons learnt and major impacts of this experiment and concludes with an outlook onto future actions.

Introduction

The Institute of Flight Mechanics of the German Aerospace Center (DLR) operates real-time systems for on-board experiments and ground support of research aircraft. In this context, the Active Control Technology / Flying Helicopter Simulator (ACT/FHS), an EC135 helicopter, is presently being modified by DLR, Eurocopter Deutschland and Liebherr Aerotechnik to enable fly-by-wire / light operation. The objective of the ACT/FHS project is to develop an airborne helicopter testbed, which will serve the various demands of research establishments, industry, and national flight test centers.

Enabling a high maintainability as well as an efficient portability to new hardware coming up during the course of the software life cycle the Process Improvement Experiment 27751 ICARUS helps to better utilise a research aircraft in terms of significantly reducing the time-to-experiment, i.e., the time needed to prepare and execute an experiment. This permanent and effective adaptation of software to different user requirements without sacrificing software quality issues is vital to achieve the DLR business objectives. In the past, the lack of appropriate specification and design processes and methods as well as the lacking tool support resulted in maintenance efforts which undermined the competitiveness of the research aircrafts in terms of costs as well as time-to-experiment.

The usage of the design tool ObjecTime [1] enables us to introduce two major process improvements: First, we are able to design and document our system and to prove the system behaviour through simulation. Second, we are enabled to achieve complete software traceability between requirements, design, and source code as well as test functions.

This project is a *Process Improvement Experiment* (PIE) funded by the Commission of the European Communities within the *European Software and Systems Initiative* (ESSI).

Baseline Project Description

The baseline project is the development of the experimental on-board equipment for a new helicopter testbed, called ACT/FHS (Active Control Technology Demonstrator and Flying Helicopter Simulator) [2]. The objective of the ACT/FHS project is the development of a technology testbed for test and validation of key technologies for future military and civil helicopters. This helicopter will be available as a multi purpose test vehicle for the evaluation of new control technologies, cockpit designs, sensor systems and man machine interfaces. The ACT/FHS project is funded by the German Aerospace Center (DLR), Eurocopter Deutschland (ECD) and the German Ministry of Defense. The test helicopter is based on the civil helicopter EC135 re-fitted with fly-by-light-technology, smart actuators, high-speed processors, intelligent sensors, and state-of-the-art display technology, Fig. 1.



Fig. 1: EC135 helicopter

The original helicopter has mechanical controls operating on hydraulic actuators. A full operative fly-by-light system is integrated, leaving the original mechanical system as a backup. This fly-by-light system - a safety critical quadruplex system - allows to fly the helicopter electronically by a safety pilot. An evaluation pilot can fly the helicopter using this system as well. The fly-by-light system does not change the pilot's feeling of the flight behaviour compared to the original unmodified helicopter. Many of the elements of the fly-by-light system are being developed by LLI (Liebherr Aerospace Lindenberg).

In the flight test mode the signals coming from the evaluation pilot's steerings are fed through an experimental computer allowing modifications of these signals before sending them to the actuators. The experimental computer is designed to realize controllers, limiters, and even simulation of the behaviour of another helicopter during flight. The experimental computer can be a simplex system. In the case of anomalies the safety pilot takes over the controls immediately.

Fig. 2 shows the overall structure of the helicopter. The safety pilot has two ways of control: the fly-by-light system and the mechanical backup. Both are certified systems to which control can be switched back for safety reasons at any time. The evaluation pilot also has two ways to control the helicopter. The first one is active when the fly-by-light control is transferred from the safety pilot to the evaluation pilot. This path usually uses a simplex experimental computer and therefore assures not the same kind of safety like the other ways of control.

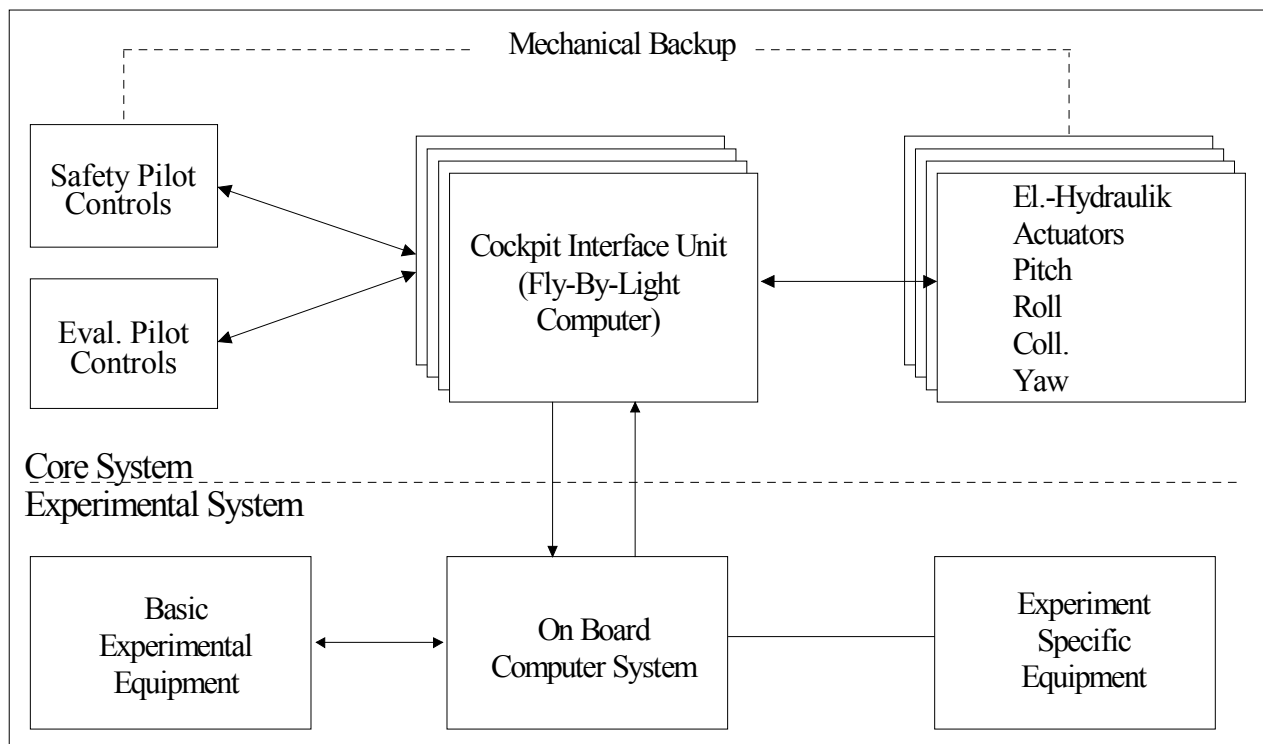


Fig. 2: ACT/FHS system overview

The experimental on-board computer system is divided into three parts (Fig. 3):

- Data Management Computer (DMC),
- Experimental Computer (EC),
- Graphic Computer (GC).

The DMC is connected to all sensors, the cockpit interface unit, the telemetry system, the data storage, the Control and Display Units (CDUs), and displays. Additionally, it is connected to the EC, and the GC. The main tasks of the DMC are to collect and distribute the sensor data, to send selected data over the telemetry to the ground station and to store selected data for later evaluation on the data storage. It can be managed during flight from the safety pilot (switches), the evaluation pilot (switches, CDU, display), and the flight test engineer (CDU, display).

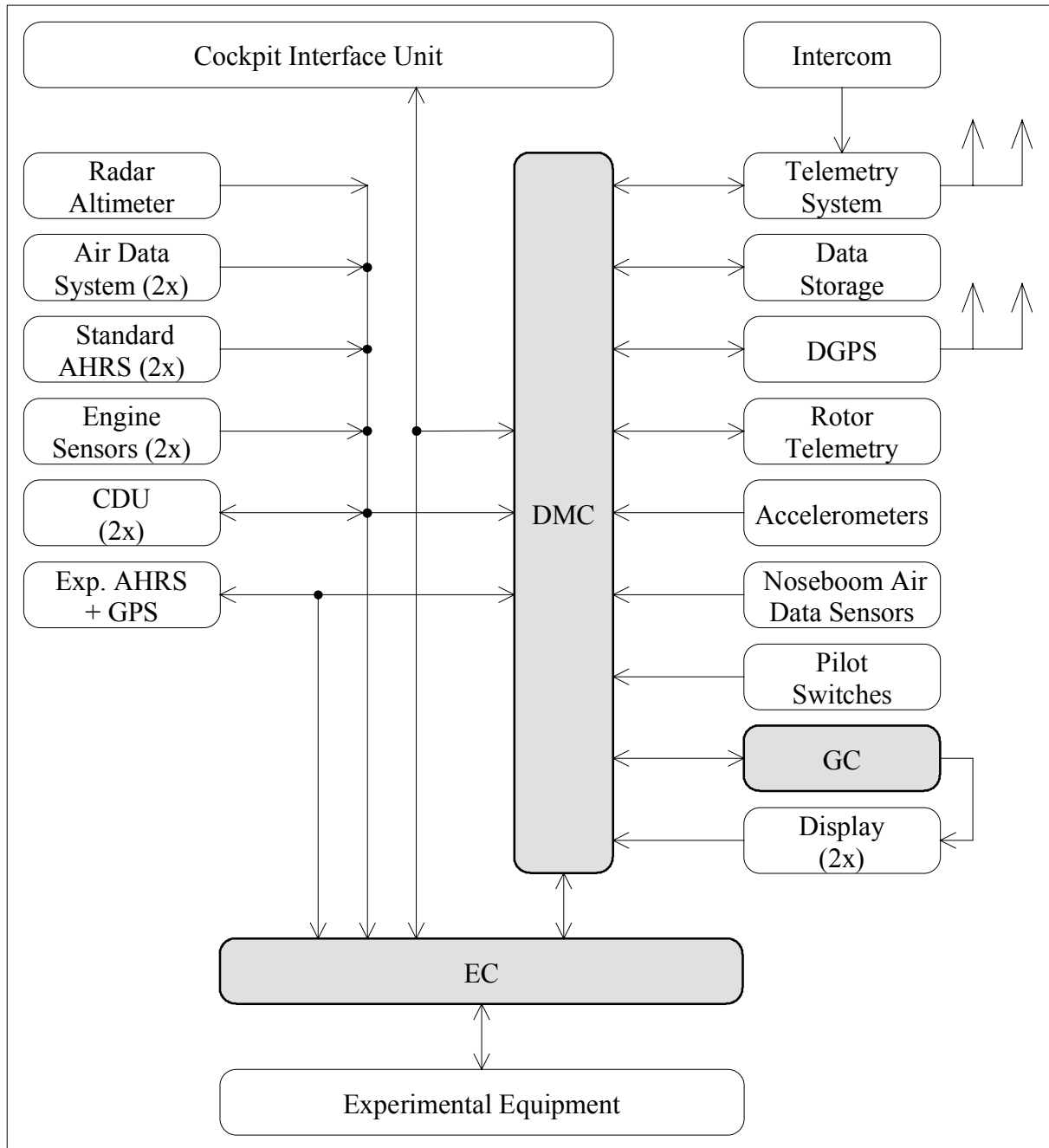


Fig. 3: On-board computer system overview

The EC is connected to the cockpit interface unit, selected sensors, the CDUs, and to optional experimental equipment. Additionally, it is connected to the DMC. The EC receives the control commands of the evaluation pilot via the Cockpit Interface Unit, calculates responses in a controller subsystem, and sends them via the Cockpit Interface Unit to the actuators of the helicopter. The DMC listens to this (ARINC-429) connection and copies the data to data storage and telemetry. The EC receives input from selected sensors to be able to use this data directly. This has been realized to allow an operation of the system without the DMC, i.e., to use an external standalone EC. The main task of the EC is to provide an interface to all kinds of present

and future experimental equipment (either directly with boards inserted into the EC or indirectly via one of the standard interface boards).

The connection between the DMC and the EC is used in the following way. The DMC sends selected sensor data (needed for the control algorithm and the experimental equipment) from the DMC to the EC. The EC sends the results and internal controller data as well as all data received from the experimental equipment to the DMC.

The GC is connected to the displays and the DMC. It drives the displays for the experimental pilot and the flight test engineer according to the data supplied by the DMC.

Process Improvement Experiment Objectives

All components of the experimental on-board computer system have to fulfil hard real-time requirements with cycle and latency times in the magnitude of 1 to 5 milliseconds and an average data throughput of about 1 MByte/s. The experimental system has to be designed to provide the user with the flexibility to modify, add, and upgrade experimental software and hardware.

While the hardware of the experimental system has to be qualified to match the requirements of flight approved systems, this does not apply to the software for the experimental system, which is - due to the safety concept of ACT/FHS - not considered to be flight critical. However, it is essential to ensure the application of best practice quality standards like the ESA [3] and DO-178B [4] software engineering standards. To adapt to the wide range of different user requirements as well as to changes in the hardware environment and infrastructure, the experimental system and especially the development of the software for the on-board computers needs sufficient flexibility to allow for frequent configuration changes and modifications, as well as system upgrades. At the same time, the experimental system is an embedded system which has to meet hard real-time requirements. This results in maintainability and portability requirements that can only be achieved via the application of a well defined software engineering approach and concept.

The objectives of the ICARUS PIE are to significantly reduce the development time and maintenance effort for this complex, embedded system via the definition and introduction of an analysis, design, and implementation process based on software best practice methods and tools as well as on the ESA [3] and DO-178B [4] software engineering standards. In addition, significant quality improvements in software verification and documentation are envisaged by the use of a corresponding design tool suitable for embedded real-time system specification and simulation.

Process Improvement Experiment Approach

The scope of the PIE is the definition and introduction of a consistent and continuous forward engineering process for the design of embedded real-time software. An appropriate tool is selected to support this engineering process. In detail, this approach to software best practice comprises the following tasks and activities within the framework of this PIE:

- Integration of the existing software engineering practices in use as well as the defined and selected software best practices standards and procedures into a consistent and continuous forward engineering process.

- Evaluation and trial application of software best practice analysis and design methods for embedded real-time systems as well as the corresponding tools.
- Provision of training and course material on the defined process and procedures as well as on the underlying software best practice standards, techniques, methods, and tools.
- Application of the defined processes, selected methodology, and toolset to the underlying baseline project.
- Improvement of the software design process, concept, and infrastructure based on the organisational and technical problems encountered during the course of the process application.
- Establishment of a process evaluation concept based on qualitative and quantitative information, collection, and assessment of selected metrics.
- Provision of quantitative information on the improvements achieved as well as collection and evaluation of qualitative statements on the applicability of the defined process.

The underlying baseline project defines the development of three software versions, called prototypes. The process improvement experiment is applied to these three phases as well. Each prototype is an upgrade of the previous and they are established successively. The following SW versions are defined within the internal ACT/FHS SW development plan:

- Prototype 1: This system is used to put interfaces into operation. This means that no sensors are connected, a basic man-machine interface is established, and a data storage is implemented.
- Prototype 2: This system is used for the basic helicopter integration as being defined within ACT/FHS. This means that a fully operable system is built including the connection of some sensors, a full man machine interface, and a full data storage.
- Prototype 3: This system will be used for handing over a fully operable helicopter to DLR users. The main difference as compared to prototype 2 is the fact that additional sensors have to be handled by the system.

In accordance with these definitions the experiment is divided into 3 phases:

- Introduction - being based upon prototype 1: Training is provided to introduce the defined process and methodology prior to applying the analysis and design on behalf of examples for prototype 1. This leads to a first acquaintance with process and methodology, being based on examples which are representative for the baseline project.
- Main application - concentrating on prototype 2: The application of the process and methodology for the prototype 2 is used to develop an embedded real-time system which has to fulfil pre-defined real-time requirements. This leads to a state of becoming familiar with process and methodology being introduced. It includes an analysis of the process improvements gained, difficulties encountered, and lessons learnt.
- Upgrade version - developing prototype 3: Improvements can be defined being based on the main application, which are going to lead to an even better application of the process and methodology for the development of the upgrade version. Since the aim of the experiment is the „introduction of computer aided analysis and design for real-time software embedded in **upgradable** systems“ additional lessons learnt will be gained since several features of prototype 3 can be considered to be upgrades of implementations for prototype 2.

Software Development Process

The requirements for the software development process determine the standards and rules for the process. The first requirements are standard ones:

- the software is developed and maintained by several people over several years;
- the software is not safety critical, however, it must be highly reliable;
- most parts of the software are real-time software with a cycle time of about 1ms.

Additionally, the following requirements must be fulfilled for the helicopter application:

- parts of the software will change significantly over the years, there is no final product;
- the experimental use will produce a tree of parallel software versions rather than a linear line of versions with one actual valid version;
- the product has to incorporate software of external companies;
- parts of the software must be changeable during flight campaigns.

The software for the on-board computer system has to provide two main services:

- A collection of system services: these are programmed once and will not change significantly in the future.
- The connection and programming of experimental equipment: these parts will change frequently.

Since the software has to be maintainable over several years and by several people, a standard software development method has to be used. However, the method has to be adapted, since no final product is delivered. Rather a flexible way of adapting to new requirements has to be used.

Although the software is not safety critical the RTCA DO-178B [4] standard for development of certified airborne systems was chosen as guideline. All procedures are observed, with the exception of writing the documents only needed for certification. Additional criteria are taken from the internal software development standards of DLR [5] and the software engineering standards of the ESA [3] for the requirements phase.

According to RTCA DO-178B the software development uses three main processes which are interlocked: The planning, development, and verification process.

During the **planning process** the subsequent methods are defined:

- quality assurance,
- configuration management,
- structure of software development,
- verification procedures,
- standards for requirements, design, and implementation.

Quality assurance uses formal approvals during each phase of the development process and a final clearance of a software version for flight tests. The configuration management allows the coexistence of several cleared software versions (for parallel projects).

The **development process** consists of five phases:

- requirements,
- design,
- implementation,
- integration,
- user documentation (i.e., the documentation needed to handle the system).

Requirements and design have to be formally approved before the next phase can start. This includes a check against the standards set up during the planning process. A final approval checks the implementation, integration, and user documentation in one step.

In order to achieve a tracing between these five phases a data base is established containing all dependencies. Containing all logical connections the data base allows automatic tracing from the requirements through the design to the code and test procedures and back again. It is expected that the effort for further upgrades and adaptations to new user requirements will be reduced by this approach.

The **verification process** uses standard verification procedures (module test, integration test, system test) to ensure high quality and the compatibility with aforementioned standards. The dependencies according to Fig. 4 are checked for completeness and traceability. The final system test uses the simulation of the EC135 helicopter at the DLR ground based simulation facility, where the software is tested within a simulated environment (helicopter, sensors, actuators, etc).

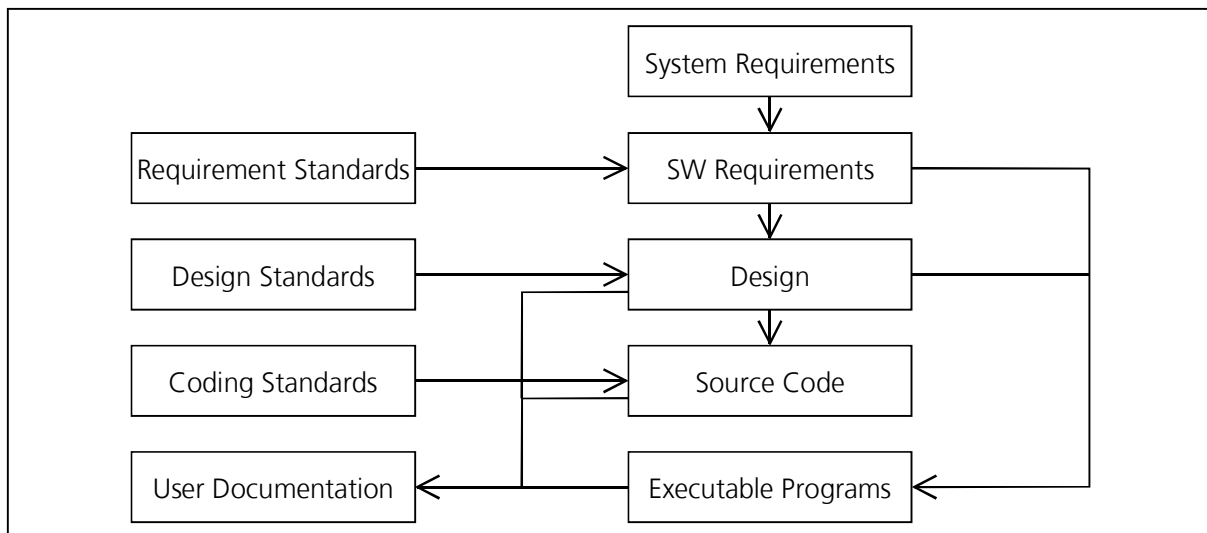


Fig. 4: Software development process and its dependencies

Traceability ensures that the links between system requirements, software requirements, design, source code, tests, and documentation are documented and maintained. The link between system requirements and software requirements is documented in the requirements document, the link between software requirements and design is documented inside the design tool, all other links are documented in the source code. All links are extracted by tools from the requirements documents, the design tool, and the source code and are collected in a trace data base.

The software verification process as well as the above described software management process do not allow „in-the-field“ source code changes. A flexible way had to be found to allow such changes during flight test campaigns: A cleared version may be changed and used without the standard verification process under the responsibility of the flight test engineer. After the campaign some or all of these software changes are incorporated back into the formal development process.

Software Development Environment

Following the hardware decision for a VMEbus based system and PowerPC processors the VxWorks real-time operating system (Wind River Systems Inc.) was chosen for most computers. The main reasons for this choice were its wide spread use and the direct support by the hardware vendor.

VxWorks offers the Tornado development environment which can be used by several developers in a networked environment under UNIX (SUN workstations) and Win95/NT (PC). The software for the target computers is programmed in C, possibly adding C++ and/or ADA components later on.

The design tool ObjecTime is used for the requirements, design and possibly the design-to-implementation phase. No special tool is used to collect and manage the requirements. The code checking tool xlint (ConSol Software GmbH) is used to check the source code against the implementation standards (usage of C, layout rules, comment rules, naming conventions, etc.).

Besides the VMEbus system a graphics computer (Silicon Graphics O2) is used to drive two flat panel (LCD) displays. Some software components are coded using the OpenGL library, instruments are designed and programmed using the tool VAPS (Virtual Prototypes Inc.). The latter allows a graphical design of instruments and the connections to data sources to animate them. It produces source code to drive the displays.

The configuration management tool CVS is used to enable history and change management of the source code and additional data files.

The trace data base is a relational SQL ORACLE data base under UNIX.

Design Tool Evaluation

In order to select the best tool and methodology for the software development, several tools were evaluated. This survey was mainly based upon a thorough study of the documentation being available from the manufacturer of each tool. Those not rejected immediately were invited to present their tool. These visits were also used to solve open questions and two times we spent a day working with the tool, in order to get a better impression of its usability. Additionally, in-house users were asked about their experience with some products. The main aspects being evaluated were:

- Interfacing with other products: Does the tool support open interfaces with other tools or standards? Especially, does the tool support our configuration management system (CVS)?
- Requirements: how are requirements handled within the tool and how are they imported from other sources?

- Design - methodology: is it suited to describe our event-driven, embedded real-time system?
- Design - checks and tests: does the tool check the design validating consistency and completeness?
- Design - libraries: are library functions supported and do they lead to restrictions?
- Design - hierarchy: is a hierarchical design possible and does this still allow checks, etc.?
- Design - drawing: how comfortable is the usage of the editor and the complete user interface?
- Code generation: does the tool support C (and what about C++, ADA)? Is it possible to include self written code and is this code affected during updates of the design?
- Connection to VxWorks: does the tool support our operating system?
- Simulation capabilities: can the design (or parts of the design) be verified through a simulation?
- Generation of documentation: which documentation standards (and templates) are supported?
- General: on which platform is the tool running and which type of licensing is supported?
- Dissemination: how many companies are using the tool, what is the expected life duration and familiarity, what can be said about possible support?

Design Tool Selection

After evaluating tools and methodologies, the tool was selected within an analytical decision making process, Tab. 1. The right most columns of the table relate to the different tools. Each row represents a requirement as it originates from the previous section. The accomplishment of each requirement was discussed and estimated for every tool according to the marks -, -, 0, +, ++ (from „very poor“ respectively „does not support“ to „very good“). Each mark is related to one of the numbers -2 to +2. Related to each requirement the second column states a weight factor which represents the importance for the evaluation according to the numbers 0 to 10 (ranging from „not important at all“ to a „very high importance“). The summarized evaluation points for each product are obtained by multiplying the weight factor for each requirement with its mark and by calculating the sum of all these specific evaluations.

Of course this table may contain subjective impressions originating from a faulty evaluation item, but nethertheless it shows that tool 3 (the product ObjecTime) seemed to be the best choice for our application. Besides ObjecTime we evaluated the tools Software through Pictures, Statemate and Teamwork. Other tools were rejected immediately because they did not fulfill some important knock-out criteria.

| Requirement | Weight | Tool 1 | | Tool 2 | | Tool 3 | | Tool 4 | |
|--------------------------|--------|--------|----|--------|----|--------|----|--------|----|
| Configuration management | 7 | + | 7 | 0 | 0 | ++ | 14 | + | 7 |
| Requirements | 5 | + | 5 | 0 | 0 | + | 5 | - | -5 |
| Design: methodology | | | | | | | | | |
| Design: checks | 9 | + | 9 | ++ | 18 | ++ | 18 | ++ | 18 |
| Design: libraries | 6 | - | -6 | 0 | 0 | + | 6 | 0 | 0 |
| Design: hierarchy | 9 | 0 | 0 | + | 9 | + | 9 | + | 9 |
| Design: drawing | 7 | + | 7 | - | -7 | + | 7 | 0 | 0 |
| Code generation: C | 3 | ++ | 6 | + | 3 | + | 3 | 0 | 0 |
| Code generation: C++ | 1 | ++ | 2 | -- | -2 | ++ | 2 | -- | -2 |

| | | | | | | | | | |
|---------------------------------|---|----|-----|---|----|----|----|----|----|
| Code generation: ADA | 1 | -- | -2 | + | 1 | -- | -2 | + | 1 |
| Code: Connection to VxWorks | 2 | -- | -4 | + | 2 | + | 2 | -- | -4 |
| Code: Reverse Engineering | 1 | + | 1 | + | 1 | -- | -2 | + | 1 |
| Code: Tests | 3 | + | 3 | - | -3 | 0 | 0 | + | 3 |
| Simulation | 8 | -- | -16 | + | 8 | ++ | 16 | + | 8 |
| Generation of documentation | 5 | + | 5 | + | 5 | 0 | 0 | + | 5 |
| User interface | 7 | - | -7 | - | -7 | 0 | 0 | - | -7 |
| Interfacing with other products | 6 | 0 | 0 | + | 6 | 0 | 0 | - | -6 |
| Platform | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Licences | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Dissemination | 4 | ++ | 8 | + | 4 | - | -4 | + | 4 |
| Life duration | 3 | + | 3 | + | 3 | 0 | 0 | + | 3 |
| Familiarity | 4 | 0 | 0 | - | -4 | - | -4 | 0 | 0 |
| Support | 3 | + | 3 | + | 3 | + | 3 | + | 3 |
| Tool integration | 6 | + | 6 | 0 | 0 | ++ | 12 | 0 | 0 |
| Evaluation Points | | | 30 | | 40 | | 85 | | 38 |

Tab. 1: Decision matrix for the tool selection

Key Lessons Learnt

The overall impression of the improved software design is considered successful. Especially due to the traceability of requirements through the design, source code and test procedures it looks promising, that the expected reduction of maintenance and upgrade effort for our embedded system will be achieved. Nevertheless the following lessons were learnt:

- According to our impression most weaknesses are seen in connection with the selected design tool. This does not mean that the tool evaluation process came to wrong conclusions considering the choice of the tool. Any tool can only be used in a successful way if it is well known by its users. We started to develop our design without having much experience with ObjecTime. We expect to improve this situation in the future.
- In the past we developed and documented our design with pencil and paper using the methodology of Hatley and Pirbhai [6]. A comparison with the past shows that the design tool does not simplify the creative thinking process, e.g., it does not encourage to try out things. Thus, sketching the design still has to be done before using the tool.
- We observed that the definition of the process as well as a thorough analysis of the requirements to be implemented simplifies design and implementation.
- Besides a well established software development process, it is worth while to spend a lot of effort investigating (and defining) the requirements since they are the basis for the software. Gathering requirements normally goes hand in hand with the so-called analysis phase. The tool ObjecTime is good in designing the software but it is not so well suited for the analysis.
- Until now we did not handle significant parts of our source code through ObjecTime. This results from the fact that we have to handle many different interfaces with external devices and we have to integrate quite a bit of „external code“ resembling the I/O drivers.

- In order to generate code for the target system automatically, the ObjecTime C version requires that all tasks (i.e., all parallel processes) are actors on the top level. Moreover, individual models have to be used for each processor. This complicates model handling significantly.
- Our system seems to lack a certain amount of “top-level communication complexity” (not enough state transitions) in order to really utilize the advantages of ObjecTime. For our system a lot of communication is hidden within specific board drivers.

Major Impacts

Today, embedded software is a major component of many technical product innovations. Thus, the ability to adapt embedded real-time software to new requirements, coming up with each new product generation, is the predominant success factor of an enterprise. This *time-to-market* pressure also applies to the airborne testbeds of DLR, since experiment specific user requirements for the embedded real-time software have to be fulfilled within tight *time-to-experiment* schedules.

From the business point of view, the marketability of the research aircraft and the corresponding experimental systems depends heavily on the cost for the preparation and performance of an experiment as well as the time-to-experiment, i.e. the time necessary to adapt the experimental system to the specific requirements of a particular customer. Since the utilisation of a research aircraft is supposed to significantly reduce the development cost and time-to-market of new aircraft, a significant reduction of these two factors is necessary to provide an attractive and competitive research aircraft and experimental system.

With the ICARUS process improvement experiment two major technical process improvements were introduced:

- The design tool ObjecTime enables us to design and document our system and to prove the system behaviour through simulation.
- We are enabled to achieve complete software traceability between requirements and source code as well as test functions. The required links from design elements to the requirements and to the source code are handled within the design tool ObjecTime.

Future Actions

Until now three main steps have been finished:

- The software process was defined, documented, and established.
- A design tool was selected.
- Introduction based on prototype 1 is completed including a design with the selected tool.

Presently we are designing the prototype 2 software. Major Future actions will be

- Design of prototypes 2 and 3 with the selected design tool, including automatic code generation.
- Further evaluation of the tool.
- Evaluating the automatic building and usage of the trace structure as shown in Fig. 4.
- Further evaluation of the complete software development process.

After the successful completion of this first process improvement step further activities focusing on the later phases of the life cycle, i.e. implementation and testing, are planned outside of this PIE.

DLR has already decided to stick to the software improvements gained within the ICARUS project, especially to continue using the design tool ObjecTime. Projects being involved in similar applications watch this process improvement very closely, a lot of discussion as well as know how transfer is already happening.

One of the most important tasks of DLR is to support the industry in sense of know how transfer. We see that the industry as well as partner research organisations are very interested in software process improvements in general and thus, there will be some future actions in this field.

Conclusions

Process improvement does not come on its own. Most important is to write down the process itself. Especially the project specific setup through the adaption of DO178B [3] was very important. The introduction of a good software development process requires quite a lot of documentation. It is important to not underestimate the effort needed. This has to be made clear for senior management. The return of investment from a good software development comes on the long run, but first it means additional effort spent.

Overall this PIE is considered very successful by the people directly involved in the ICARUS work and by the management of the baseline project as well as by superiors. Until now weaknesses are only seen in connection with the selected design tool. An evaluation of these weaknesses lead to the conclusion that:

- It has to be considered fairly normal to detect weaknesses with tools.
- One needs to become an expert with a tool before one knows and avoids all disadvantages.

Nevertheless one should keep in mind that the definition of the software development process itself as well as a thorough analysis of the requirements to be implemented seem to be much more efficient than trying out things with a design tool.

Glossary

| | |
|----------|--|
| ACT/FHS: | Active Control Technology / Flying Helicopter Simulator |
| AHRS: | Attitude and Heading Reference System |
| CDU: | Control and Display Units |
| DLR: | Deutsches Zentrum für Luft- und Raumfahrt e.V. |
| DMC: | Data Management Computer |
| EC: | Experimental Computer |
| ECD: | Eurocopter Deutschland |
| ESA: | European Space Agency |
| ESSI: | European Software and Systems Initiative |
| GC: | Graphic Computer |
| GPS: | Global Positioning System |
| ICARUS: | Introduction of Computer Aided Analysis and Design for Real-Time Software Embedded in Upgradable Systems |

LLI: Liebherr Aerospace Lindenberg
PIE: Process Improvement Experiment
ROOM: Real-Time Object-Oriented Modeling
SW: Software

References

- [1] Real-Time Object-Oriented Modeling. B. Selic, G. Gullekson, P. Ward, John Wiley & Sons, New York, 1994.
- [2] ACT/FHS On Board Computer System. K. Alvermann, R. Gandert, B. Gelhaar, S. Graeber, H. Oertel, European Telemetry Conference, Garmisch Partenkirchen, 1998.
- [3] Software Engineering Standards: European Space Agency. ESA PSS-05-0, Issue 2, Februar 1991.
- [4] Software Considerations in Airborne Systems and Equipment Certification. RTCA DO-178B, Dec. 1992.
- [5] Software Quality Standards für kleine Projekte, DLR, Hauptabteilung Qualität und Sicherheit, April 1997.
- [6] Strategies for Real-Time System Specification. D. Hatley, I. Pirbhai, Dorset House Publishing, 1988.

Developing Embedded Software for a Helicopter Testbed

Institute of Flight Mechanics
Deutsches Zentrum für Luft- und Raumfahrt e.V.
Braunschweig, Germany



Klaus Alvermann
Stephan Graeber
Henrik Oertel
Lothar Thiel



3.11.99

QWE '99



Contents

- Helicopter Baseline Project
- Process Improvement Experiment
 - Objectives
 - Approach
- Defined SW Development Process
- Software Development Environment
- Design Tool Selection
- ROOM - Object Time Modeling
- Lessons Learnt / Future Plans
- Conclusions

3.11.99

QWE '99

2



Background / Baseline Project

- **Goal:**
 - Operation of an experimental FBW helicopter
- **Motivation:**
 - Testbed for in-flight simulation & technology demonstrator
- **Customers (European):**
 - Industry
 - Ministry of Defense
 - Research Establishments
 - Pilot schools

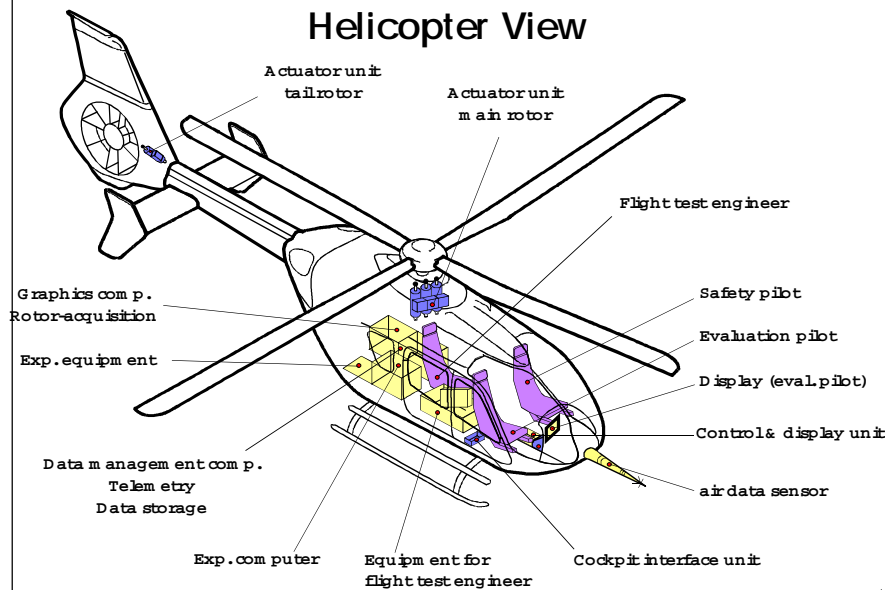
3.11.99

QWE '99

3



Helicopter View

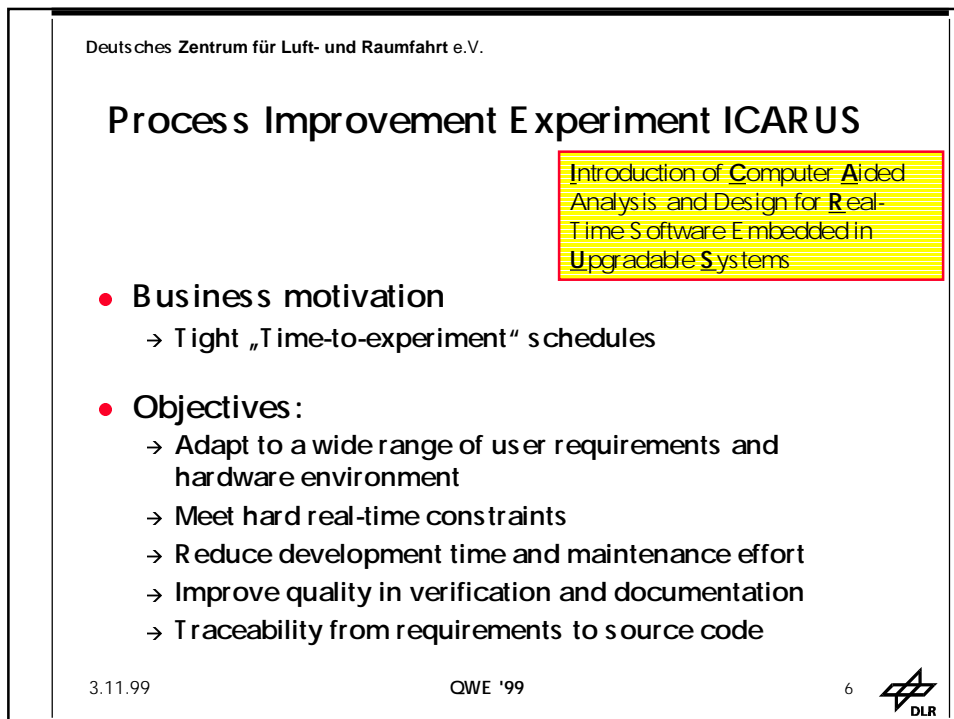
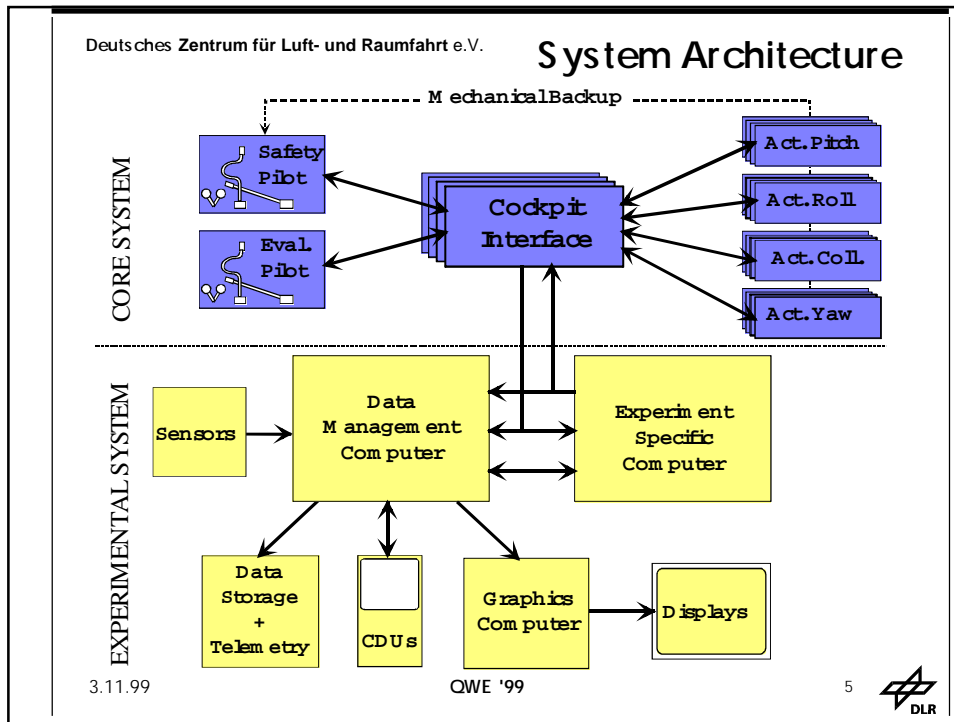


3.11.99

QWE '99

4





PIE: Approach

- Integrate a consistent engineering process
- Evaluate SW best practice design methods for embedded real-time systems
- Provide training on the defined process
- Improve software design process, concept, and infrastructure
- Establish a process evaluation concept
- Adapt to software versions of baseline project
 - Prototype 1: Introduction
 - Prototype 2: Main application
 - Prototype 3: Upgrade version

3.11.99

QWE '99

7



Software Development Processes

Guidelines: RTCA/DO-178B
ESA Software Engineering Standards
Company SW Development Standards

- Planning Process
- Development Process
 - Requirements
 - Design
 - Implementation
 - Integration
- Control Process
 - Verification
 - Configuration Management
 - Quality Assurance

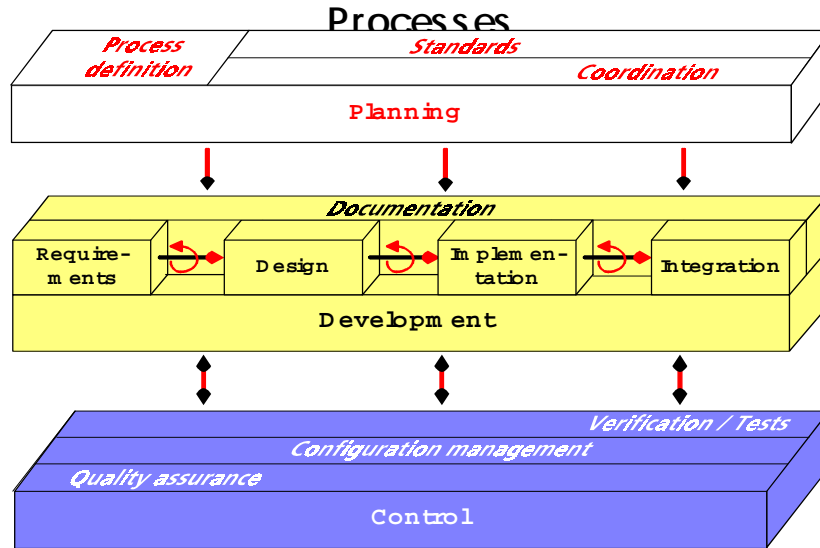
3.11.99

QWE '99

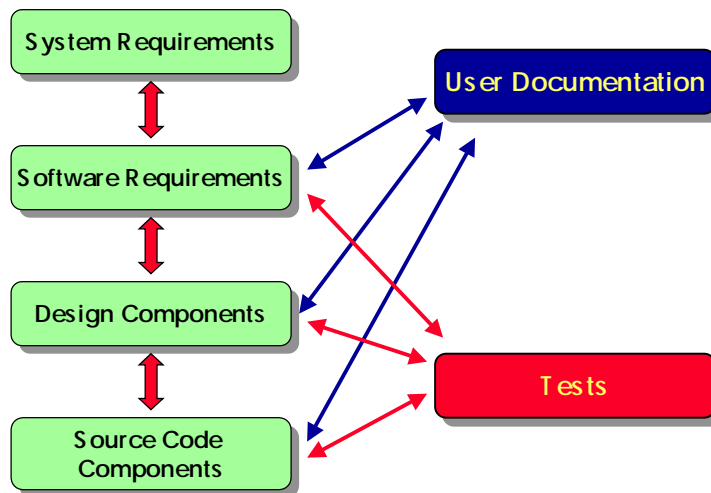
8



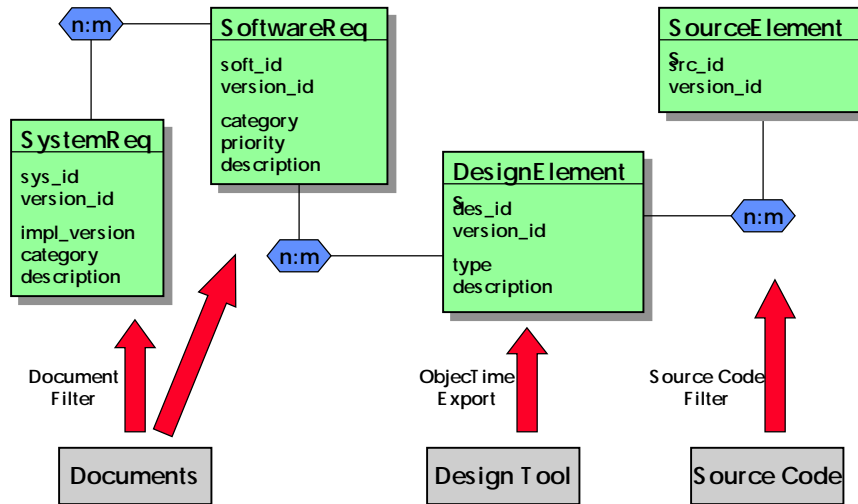
Controlled Software Development



Verification: Traceability



Trace Database



3.11.99

QWE '99

11



Software Development Environment

- VxWorks real-time operating system (VME bus)
- Object time design tool
- VAPS instrument design tool (OpenGL interface)
- C programming language
- CVS configuration management tool
- xlint source code checker
- ORACLE trace data base
- Helicopter simulation for HIL - Tests

3.11.99

QWE '99

12



Design Tool Evaluation

- Interfacing with other products (e.g. CVS)?
- Handling of requirements?
- Is the methodology suited for a real-time system?
- Checks for completeness and consistency?
- Are library functions supported?
- Is a hierarchical approach supported?
- Drawings - how comfortable is the editor?
- Support of code generation (C), ((C++)), (((ADA)))?)
- Connection to VxWorks?
- Simulation capabilities?
- Documentation?

Design Tool Selection

| Requirement | Weight | Tool 1 | Tool 2 | Tool 3 | Tool 4 |
|---------------------------------|--------|--------|--------|--------|--------|
| Configuration management | 7 | + 7 | 0 0 | ++ 14 | + 7 |
| Requirements | 5 | + 5 | 0 0 | + 5 | - -5 |
| Design: methodology | | | | | |
| Design: checks | 9 | + 9 | ++ 18 | ++ 18 | ++ 18 |
| Design: libraries | 6 | - -6 | 0 0 | + 6 | 0 0 |
| Design: hierarchy | 9 | 0 0 | + 9 | + 9 | + 9 |
| Design: drawing | 7 | + 7 | - -7 | + 7 | 0 0 |
| Code generation: C | 3 | ++ 6 | + 3 | + 3 | 0 0 |
| Code generation: C++ | 1 | ++ 2 | -- -2 | ++ 2 | -- -2 |
| Code generation: ADA | 1 | -- -2 | + 1 | -- -2 | + 1 |
| Code: Connection to VxWorks | 2 | -- -4 | + 2 | + 2 | -- -4 |
| Code: Reverse Engineering | 1 | + 1 | + 1 | -- -2 | + 1 |
| Code: Tests | 3 | + 3 | - -3 | 0 0 | + 3 |
| Simulation | 8 | -- -16 | + 8 | ++ 16 | + 8 |
| Generation of documentation | 5 | + 5 | + 5 | 0 0 | + 5 |
| User interface | 7 | - -7 | - -7 | 0 0 | - -7 |
| Interfacing with other products | 6 | 0 0 | + 6 | 0 0 | - -6 |
| Platform | 2 | 0 0 | 0 0 | 0 0 | 0 0 |
| Licences | 2 | 0 0 | 0 0 | 0 0 | 0 0 |
| Dissemination | 4 | ++ 8 | + 4 | - -4 | + 4 |
| Life duration | 3 | + 3 | + 3 | 0 0 | + 3 |
| Familiarity | 4 | 0 0 | - -4 | - -4 | 0 0 |
| Support | 3 | + 3 | + 3 | + 3 | + 3 |
| Tool integration | 6 | + 6 | 0 0 | ++ 12 | 0 0 |
| Evaluation Points | | 30 | 40 | 85 | 38 |

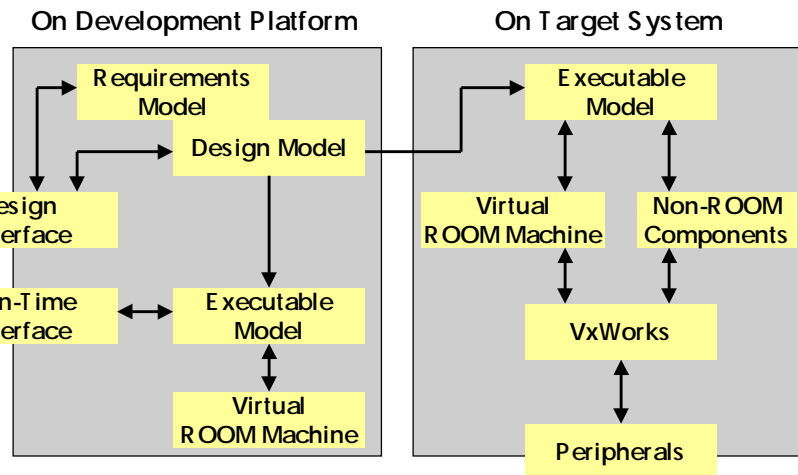
ROOM Methodology

- **Timeliness** reaction within pre-scribed interval
- **Dynamics** structure changes during run-time
- **Reactiveness** system is event driven
- **Concurrency** multiple concurrent threads
- **Distribution** multi-processor architecture

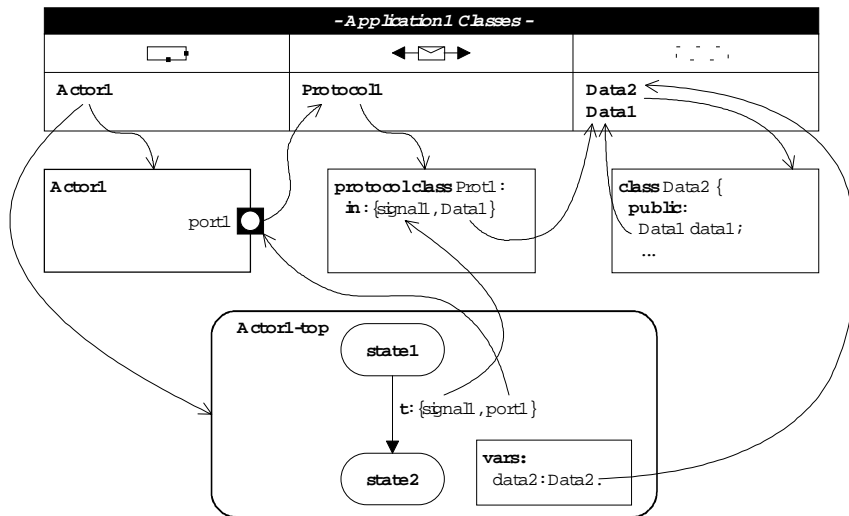
Descriptions are based on

- Abstraction
- Completeness
- Hierarchy
- Incremental modeling
- Reusability

Simulation



Overview of ROOM Modeling Language



3.11.99

QWE '99

17



Key Lessons Learnt

Overall impression is considered successful, nevertheless:

- We need more experience with the tool
- Sketching of the design still has to be done
- A thorough analysis simplifies design and implementation
- Object time induces a slightly different development approach, conflicting with our analysis phase
- Support for external hardware cannot be handled within a design tool

3.11.99

QWE '99

18



Software for Prototype 1 is completed

Future Plans

- Design of further SW prototypes, including automatic code generation
- Further usage (evaluation) of the tool
- Further improvement of software development process
- Dissemination of results, internal and external

Conclusions

- **Technical Impacts:**
 - Design and documentation of our system
 - Proving behaviour through simulation
 - Complete traceability (between requirements, design, source code, test functions)
- **Business Impact:**
 - Experiment specific user requirements for embedded real-time software can be fulfilled within tight *time-to-experiment* schedules
- **Key Success Factors:**
 - The process itself is most important
 - Thorough analysis of requirements helps a lot

Quality Week Europe 1999

Engineering Safety Related Control Software in Developing Countries

Wolfgang A. Halang
FernUniversität
Fachbereich Elektrotechnik
D-58084 Hagen

Matjaž Colnarič
University of Maribor
Faculty of Electrical Engineering and Computer Science
SLO-2000 Maribor

Boundary Conditions

- Decline of hardware costs - rapid growth of software costs; reasons: lower degree of automation of "software engineering" and high labour costs.
- Developing countries often have many highly qualified young people, who may work in software development.
- For developing countries it is economically feasible to concentrate on software development:
 - lower initial investments,
 - big demand for software,
 - sw-market larger than hw-market,
 - greater added value.

2

The Niche

Problem of geographical distance to customers in industrialised countries:

- low chance for customer specific software specified orally “on the fly”,
- concentration on standard software and libraries is necessary.

Industrial process automation is an important market for standard software.

The necessity of safety licensing for safety related applications

- creates much work for people and
- high added value.

3

Programming Paradigm

Restriction to application domains with software of limited variability and clearly defined functionality: industrial process automation.

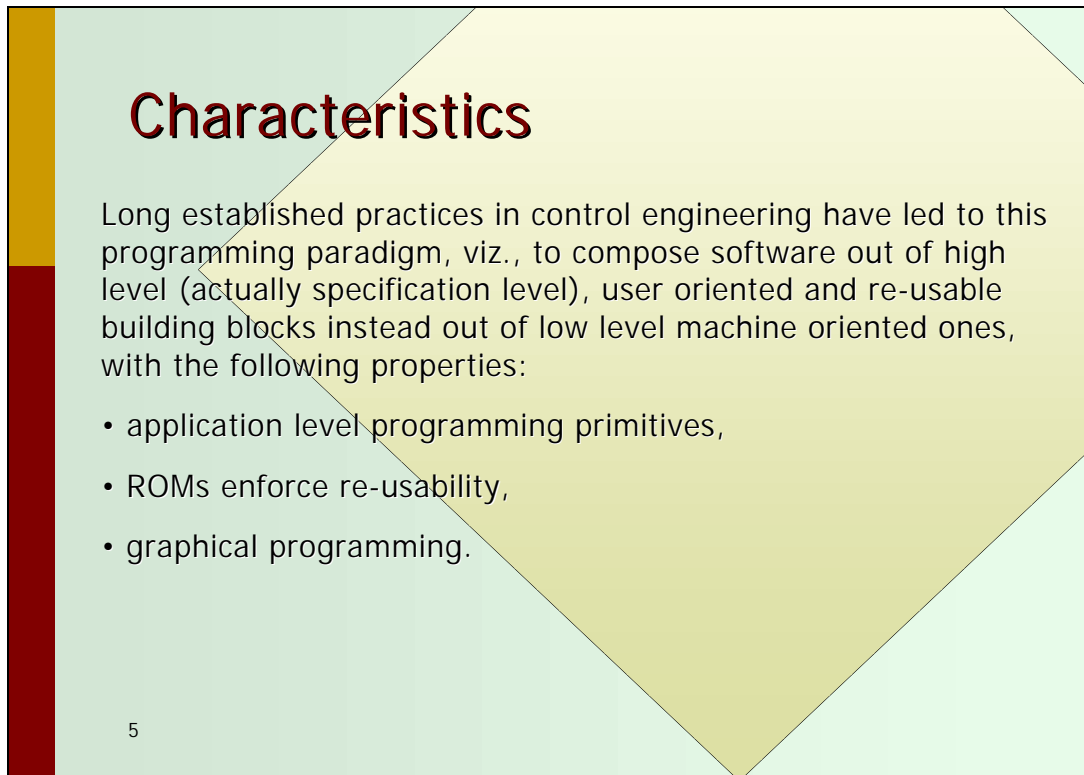
Identification and set-up of small libraries (max. 80) of standardised (e.g., VDI/VDE 3696) and rigorously verified, application specific basic functions/blocks (for a larger closed problem area each), and provision of corresponding machine code in (E)(P)ROMs.

Graphical “wiring” of function (block) instances in FBD/SFC of IEC 1131-3:

- programming-in-the-large,
- quality of specification.

Verification by “diverse back translation”

4



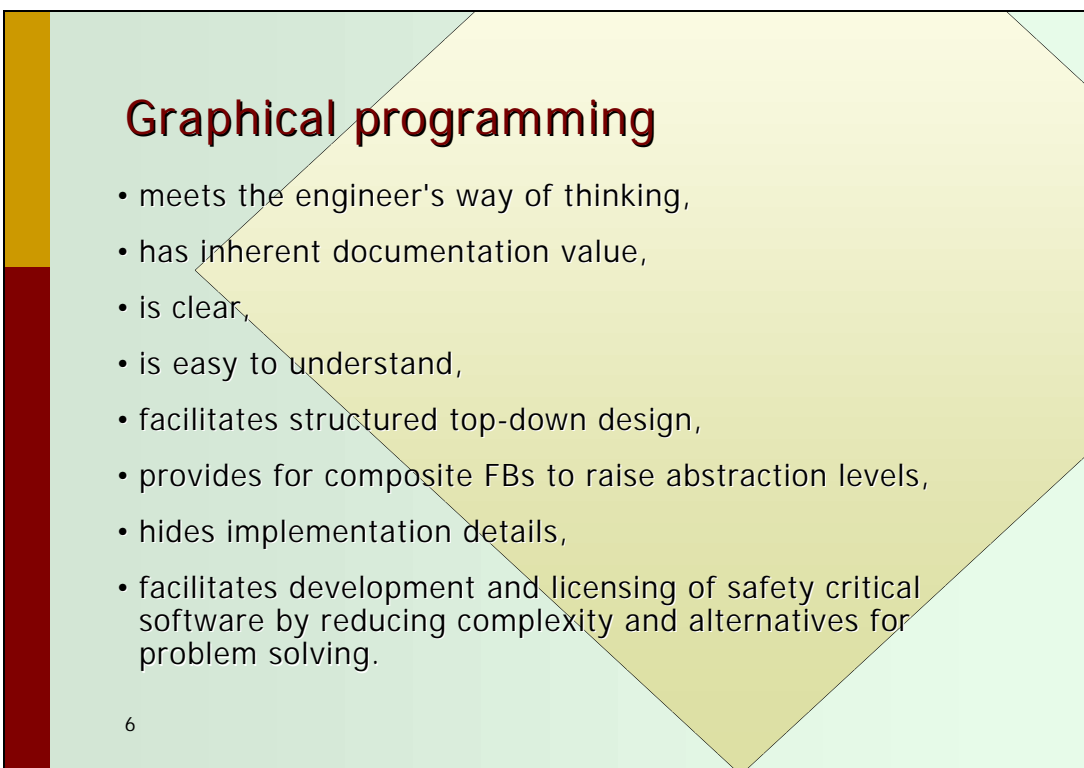
Characteristics

Long established practices in control engineering have led to this programming paradigm, viz., to compose software out of high level (actually specification level), user oriented and re-usable building blocks instead out of low level machine oriented ones, with the following properties:

- application level programming primitives,
- ROMs enforce re-usability,
- graphical programming.

5

The slide features a decorative background with a vertical bar on the left side, split into yellow and red segments, and a large yellow diamond shape in the center. The text is positioned to the left of the diamond.



Graphical programming

- meets the engineer's way of thinking,
- has inherent documentation value,
- is clear,
- is easy to understand,
- facilitates structured top-down design,
- provides for composite FBs to raise abstraction levels,
- hides implementation details,
- facilitates development and licensing of safety critical software by reducing complexity and alternatives for problem solving.

6

The slide features a decorative background with a vertical bar on the left side, split into yellow and red segments, and a large yellow diamond shape in the center. The text is positioned to the left of the diamond.

The Language Standard IEC 1131-3

defines a family of 4 mutually transformable languages:

IL Instruction List (Assembly language)

LD Ladder Diagram (Formalisation of electrical circuit diagrams to describe relay based binary controls)

FBD Function Block Diagram

ST Structured Text

as well as

SFC Sequential Function Chart

IEC 1131-3 Language FBD

FBD is signal-flow oriented and design-object oriented.

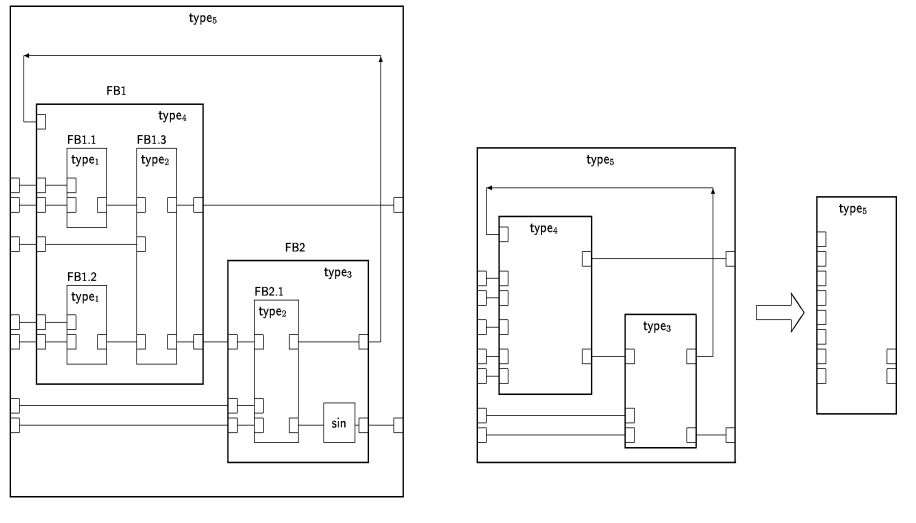
FBD's sole elements are

- function (block) instances,
- connectors and connecting lines,
- names,
- tasks.

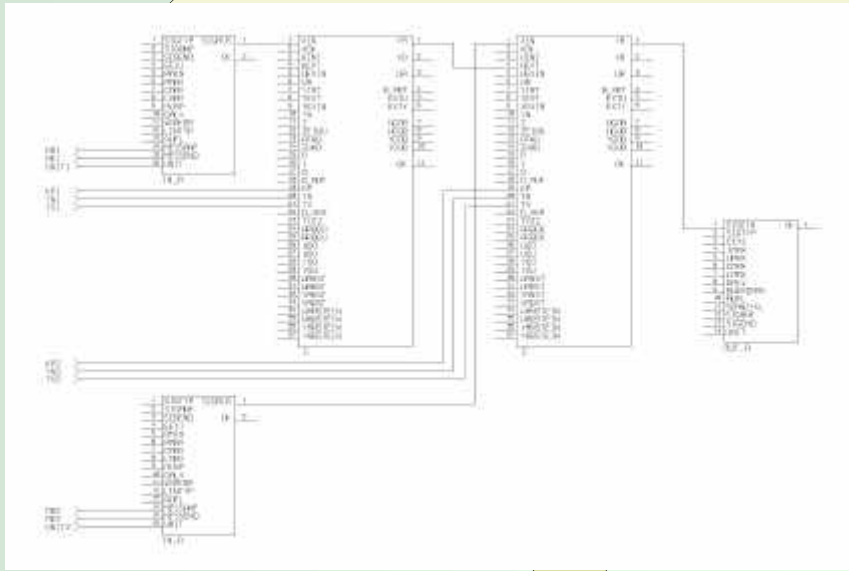
Advantages:

- Specifications are directly mapped onto sequences of procedure calls.
- Software complexity is reduced by orders of magnitude,
- Compiled object code contains procedure calls and internal moves of data, only.

Composite Function Blocks



A Function Block Diagram



IEC 1131-3 Language SFC

SFC is activity-flow oriented.

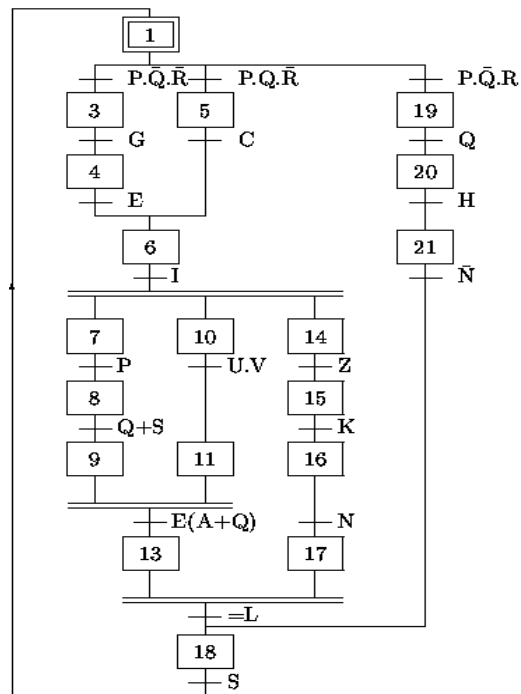
SFC's sole elements are

- (initial) steps,
- transitions,
- actions.

SFC (Grafcet) is the industrial implementation of the Petri net concept.

When a process is within a certain state (step), the sequential function chart describes the system reaction in dependence on the value of the subsequent transition condition.

A Sequential Function Chart



Guideline VDI/VDE 3696

“Manufacturer independent configuration of digital control systems”

67 modules from the following function (block) classes:

- Monadic mathematical functions
- Polyadic mathematical functions
- Comparisons
- Monadic Boolean function
- Polyadic Boolean functions
- Edge detectors
- Selection functions
- Counters, monostables, bistables, timers
- Process input/output
- Network communication input/output
- Dynamic elements and regulators
- Conditioning for display and operation

Software Safety Licensing

The two-step development procedure propagates into the verification phase:

1. Before being released, a library of function blocks is proven correct - once for all with appropriate rigorous (formal) methods.
2. For a given application, only the interconnection of instantiated function blocks (i.e., a certain data flow) needs verification.

This is performed by diverse back translation of corresponding object programs on the level of module invocations.

Verifying Function Block Libraries

This is feasible due to the limited complexity and size (≤ 2 pages) of standardised basic function blocks.

Their correctness can and must be formally proven with bearable effort and mathematical rigour.

However, this can be carried through by specialists, only.

The costly safety licensing is required only once in the lifetime of a function block library.

These costs are justified by the safety requirements, and can be spread over many implementations.

Formal Verification

Applicable methods are:

- complete test (sometimes),
- symbolic evaluation,
- predicate calculus (Dijkstra, Hoare),
- Z,
- HOL/Isabelle,
- whatever appropriate.

Automated tools may often be used.

Often the re-use of function blocks allows to re-use corresponding proofs.

Diverse Back Translation

- is assumed to be the most powerful, the only generally applicable, and the only officially recognised (and developed) - by the licensing authorities (TÜV) - method for software verification;
- consists in re-gaining a requirement specification by several, independently working licensor groups on the basis of loaded and read-out machine code, and granting a license upon equivalence;
- not feasible for most industrial applications.

17

Diverse Back Translation

- is, however, economical feasible on the level of graphical FBD interconnections, where programs are shorter and simpler by orders of magnitude, and where one easy transformation step leads back from machine code to problem specification.
- Advantages: essentially informal, but rigorous, commonly understandable (law suits), and immediately applicable without any training.
- Diversity can be provided with one human licensor only.
- Diverse back translation elegantly circumvents the problem of non-safety-licensed compilers.

18

Work to be Outsourced

- Analysis of an application area,
- Definition and validation of an appropriate function (block) library,
- Verification of specifications,
- Programming of modules,
- Verification on source language level ,
- Verification on object code level by diverse back translation,
- Tool development,
- Documenting all products and phases,
- In principle also: licensing.

19

Work in Industrialised Countries

- Graphical construction of application specific software in a dialogue with the customer, who has usually described his problem already in form of diagrams.
- Verification of the calling sequences of instantiated software modules by diverse back translation.
- Diversity provided by personal and spatial separation of the construction and verification steps.

20

Concept of a Software House

Very low initial investments:

- Office space,
- Workstations, PCs,
- Internet connection,
- Software tools (often in PD).

Highly qualified staff:

Linking to a university institute feasible according to the " spin-off " model.

21

Conclusion

Program controlled safety related systems are an economical must.

The problem of software dependability will further exacerbate severely, particularly in process automation.

A software engineering method was presented here, which meets societies' high dependability requirements for technical systems.


Through outsourcing, developing countries can achieve high added value and good exports with low investments. They can thus create urgently needed jobs for people, who otherwise may emigrate.

As a side effect, technology is transferred.

22

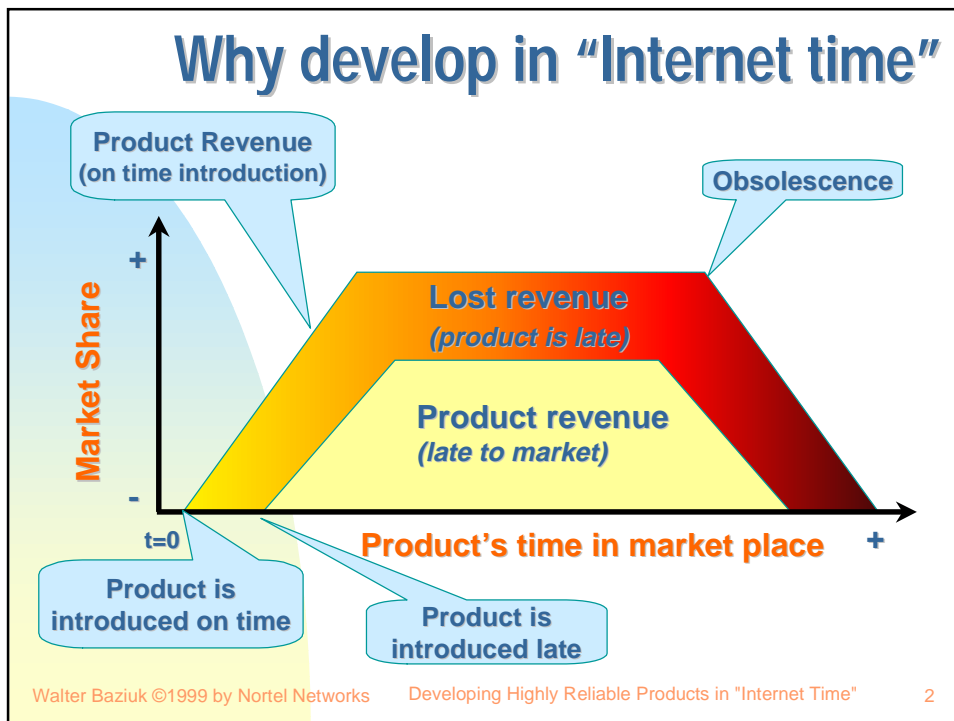

Developing Highly Reliable Products in "Internet Time"

Applications/
Solutions



Track
5A

Walter Baziuk
baziuk@nortelnetworks.com
QWE99 - Brussels, Belgium - November 3, 1999

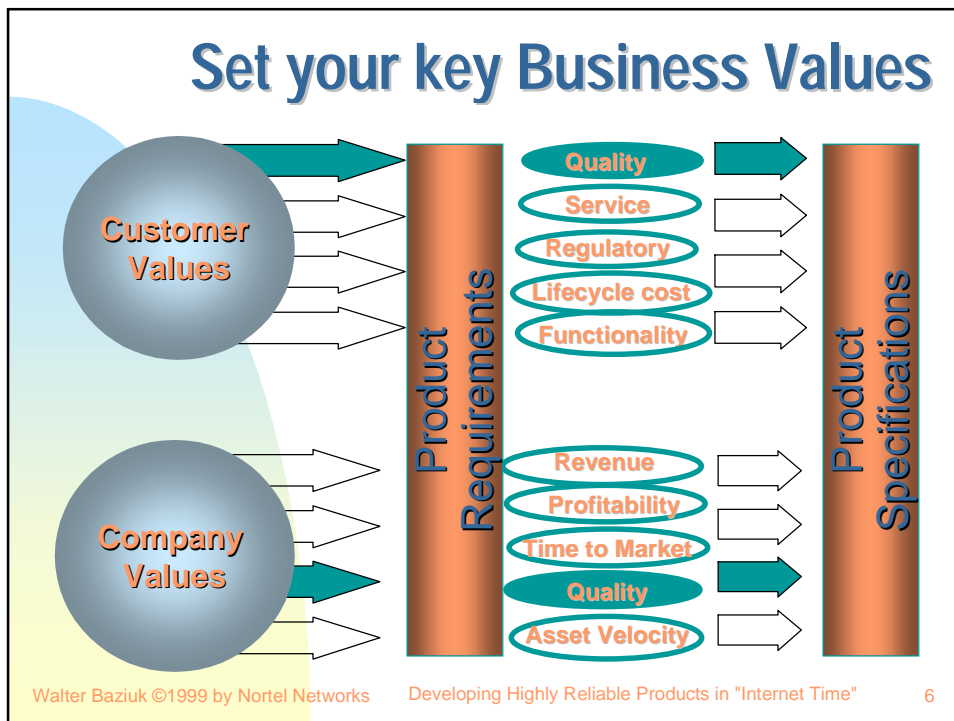
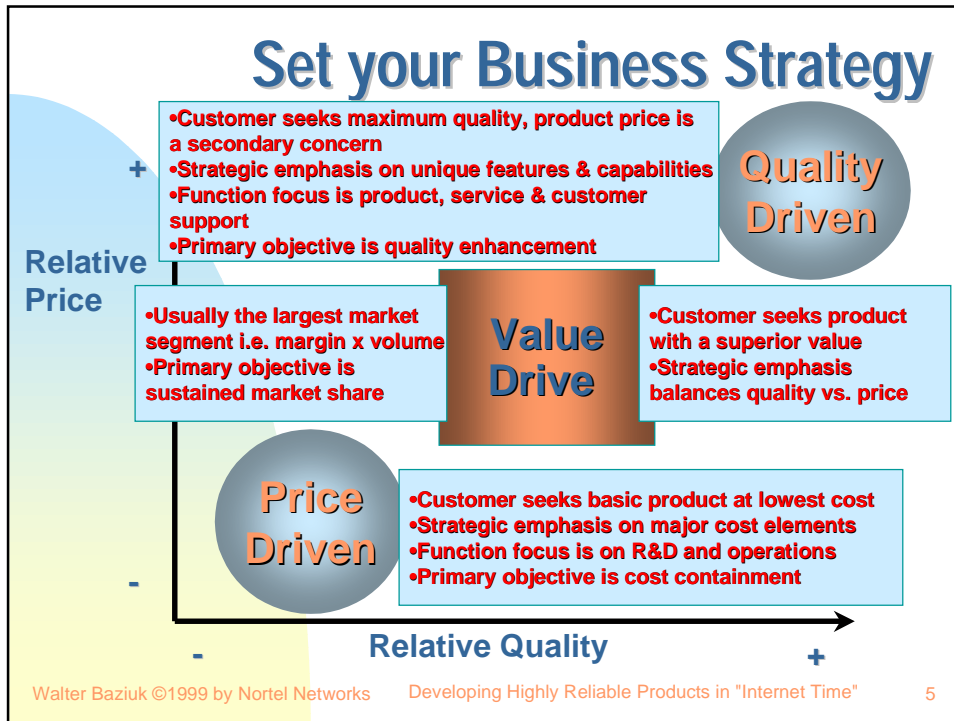


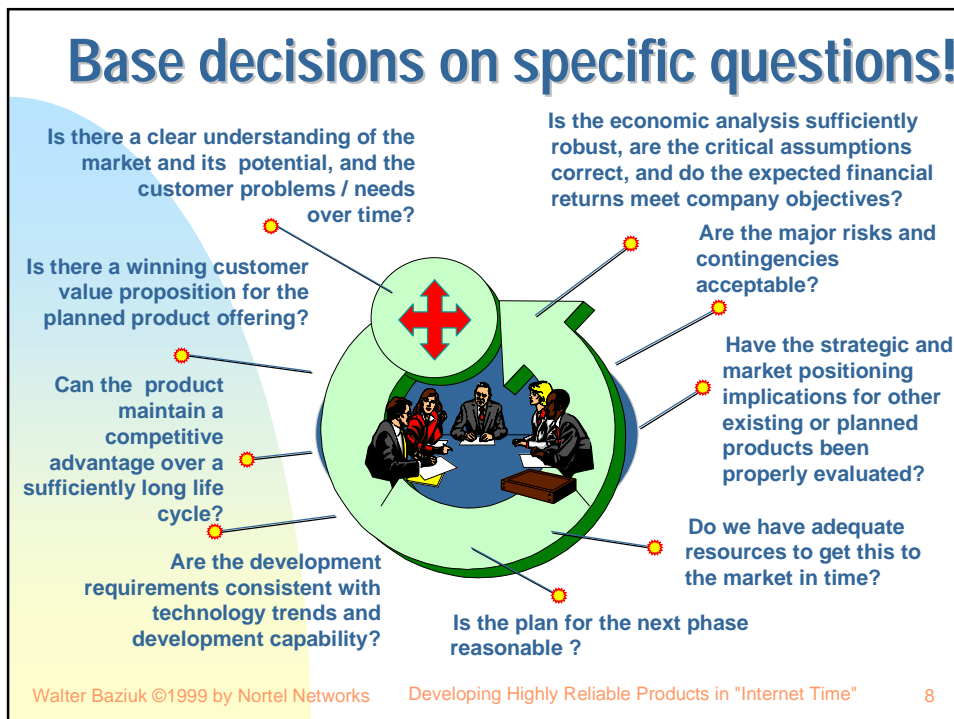
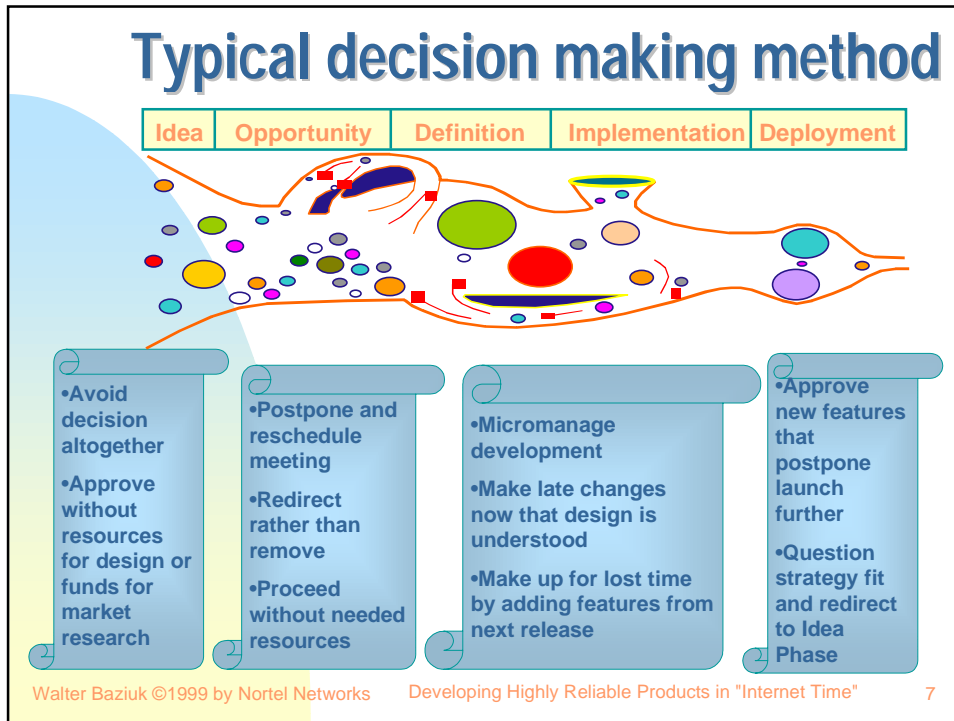
Agenda

- ▶ **Identify key market/customer requirements**
 - ◆ Work with your customer determine "which features are in and which aren't" in each release.
 - ◆ Manage requirements churn and still deliver on time.
 - ◆ Get the customer involved earlier and have them witness intermediate results.
- **Manage product reliability and quality**
- **Accelerate software development**

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 3








Key time to market enablers

Customer Focused Design



What we deliver vs. what the customer expects


Shareholder value metrics:

Adaptability:
% Utilization

Efficiency:
Cost of cancellations

Effectiveness:
TTM /TTP


Viability business case



- Market Plan
- Corporate Strategy
- Business Plan
- Technical Plan
- Customer Focused Design

End-to-End Accountability:

- Schedule Commitments
- Budget Commitments
- Specification Commitment
- People Commitments




Inputs
from
Project
Metrics

Outputs
to
Business
Metrics

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 9

Decisions making fundamentals



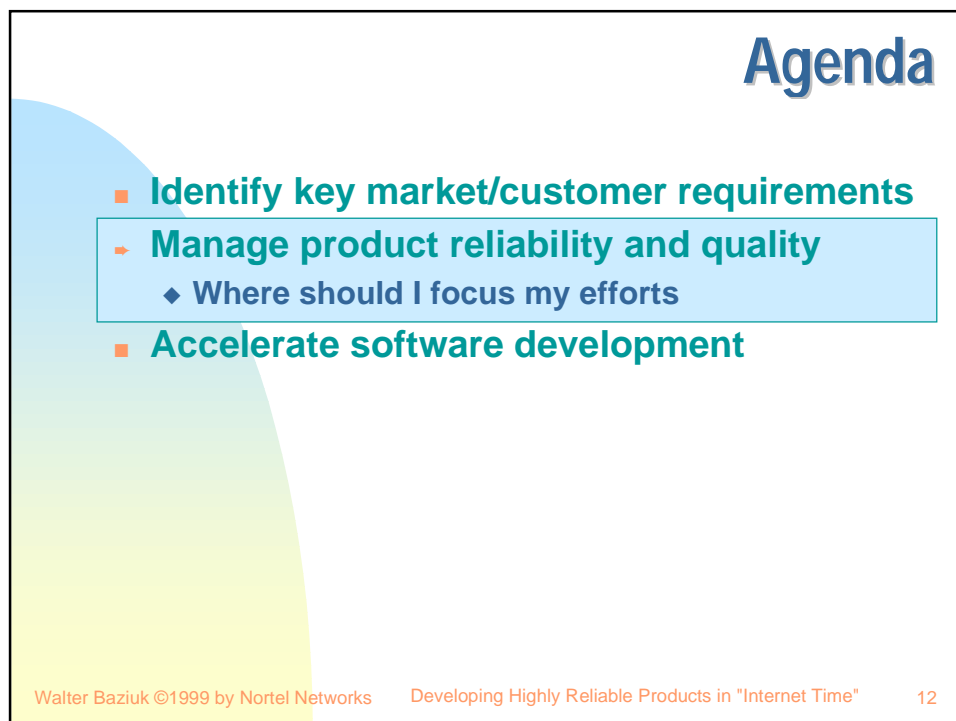
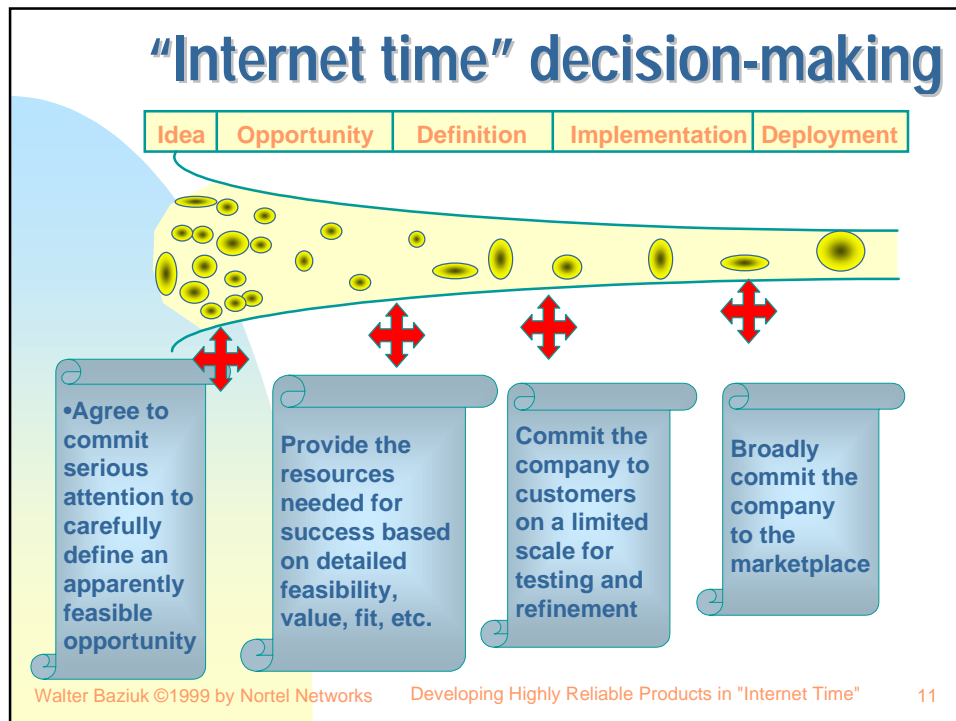
A real authority must make the decisions

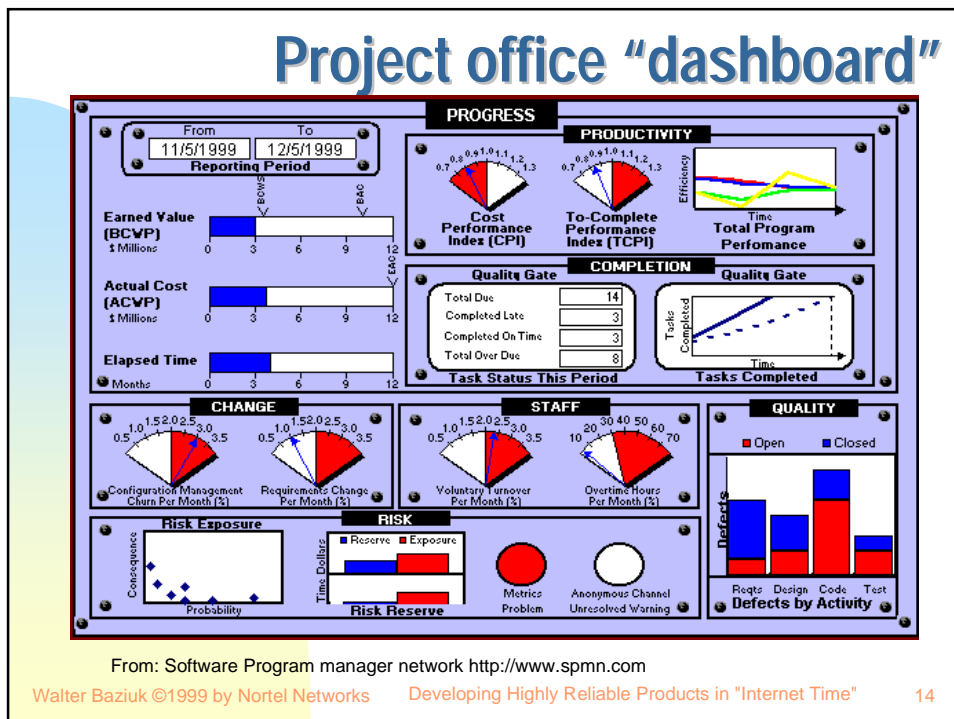
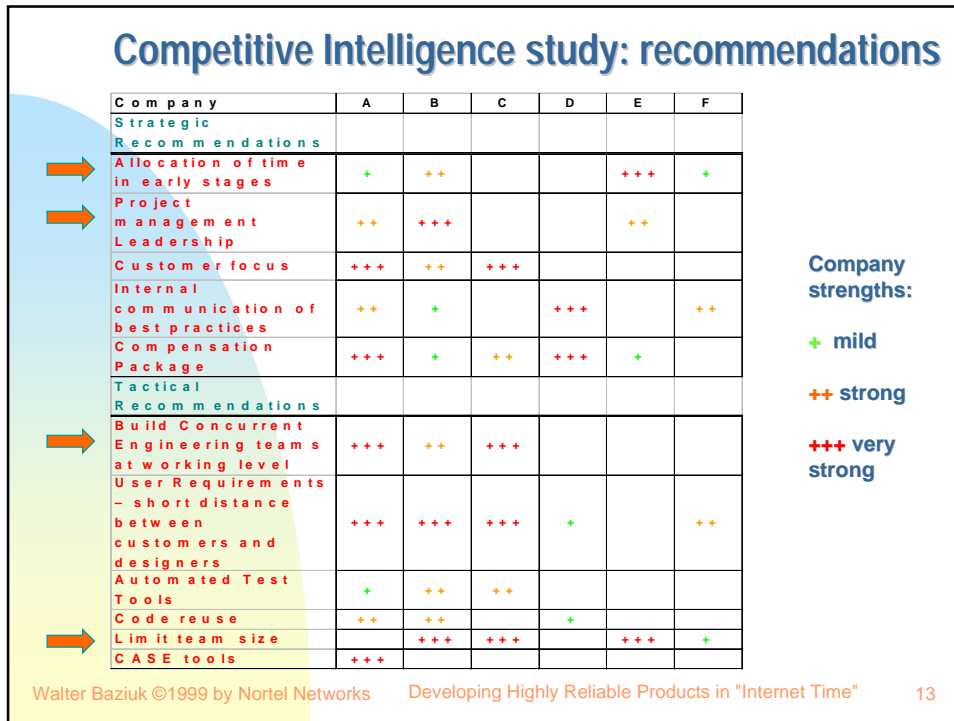
Unambiguous decisions must result

“Internet time” delivery demands making decisions that stick

The decision maker must have authority over the product domain

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 10





Fault Classification: trend vs. phase

| Product Phase | Activity / Source of Fault | Faults / KLOC | % contribution | Faults per phase | Cumulative Faults |
|---------------------|--------------------------------|---------------|----------------|------------------|-------------------|
| Requirements | Product Requirements Document | 2.5 | 5.0% | 5% | 5% |
| Definition | Product Objective Document | 3.5 | 7.0% | 24% | 29% |
| | Product Specification Document | 8.5 | 17.0% | | |
| Design | Architecture Inspection | 5.0 | 10.0% | 25% | 54% |
| | Design Inspection | 7.5 | 15.0% | | |
| Coding | Coding | 6.4 | 12.8% | 13% | 67% |
| Development testing | Module Test plan | 0.5 | 1.0% | 20% | 87% |
| | Module Test | 4.5 | 9.0% | | |
| | Unit Test plan | 0.5 | 1.0% | | |
| | Unit Test | 4.5 | 9.0% | | |
| Integration testing | Functional Test plan | 0.5 | 1.0% | 13% | 100% |
| | Functional Test | 3.5 | 7.0% | | |
| | System Test plan | 0.3 | 0.6% | | |
| | System Test | 2.0 | 4.0% | | |
| | Regression Test plan | 0.0 | 0.0% | | |
| Post Delivery | Regression Test | 0.2 | 0.4% | | |
| | Customer Problem reports | 0.1 | 0.2% | 13% | 100% |
| Total | | 53 | | | 100% |

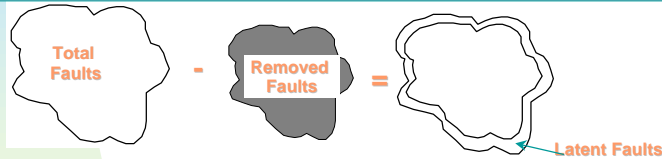
Faults/KLOC = (Major faults + (Minor faults/ 3) / Total KLOC for product
 Major faults = Service affecting problem
 Minor faults = Non-service affecting problem

Latent Faults

If we manufactured 100,000 light bulbs, sampled 1000 and found 42 to be defective;

4200

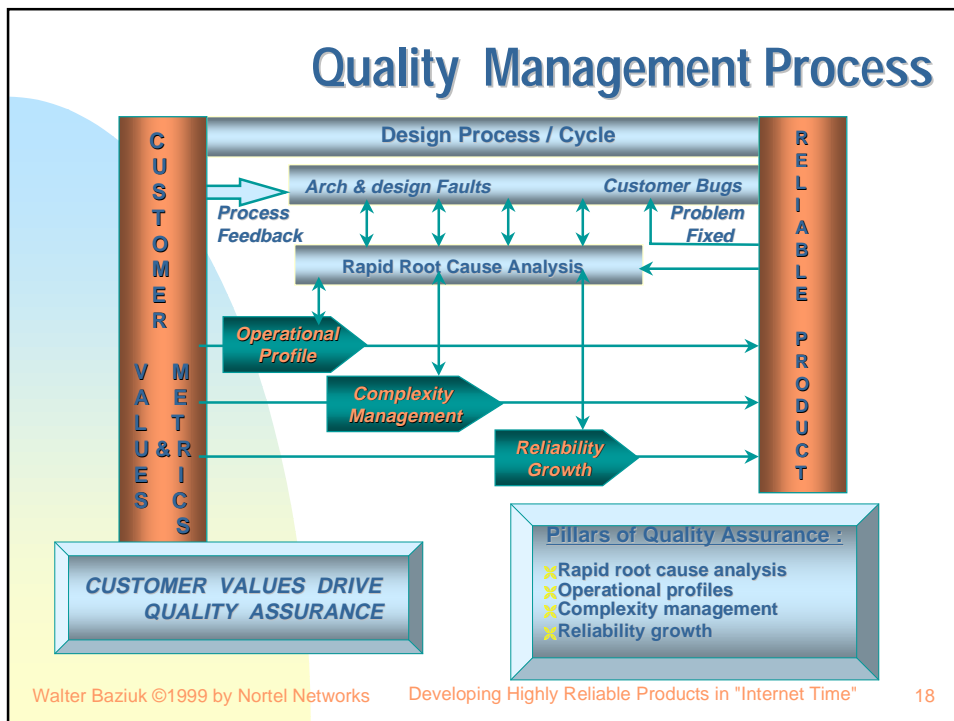
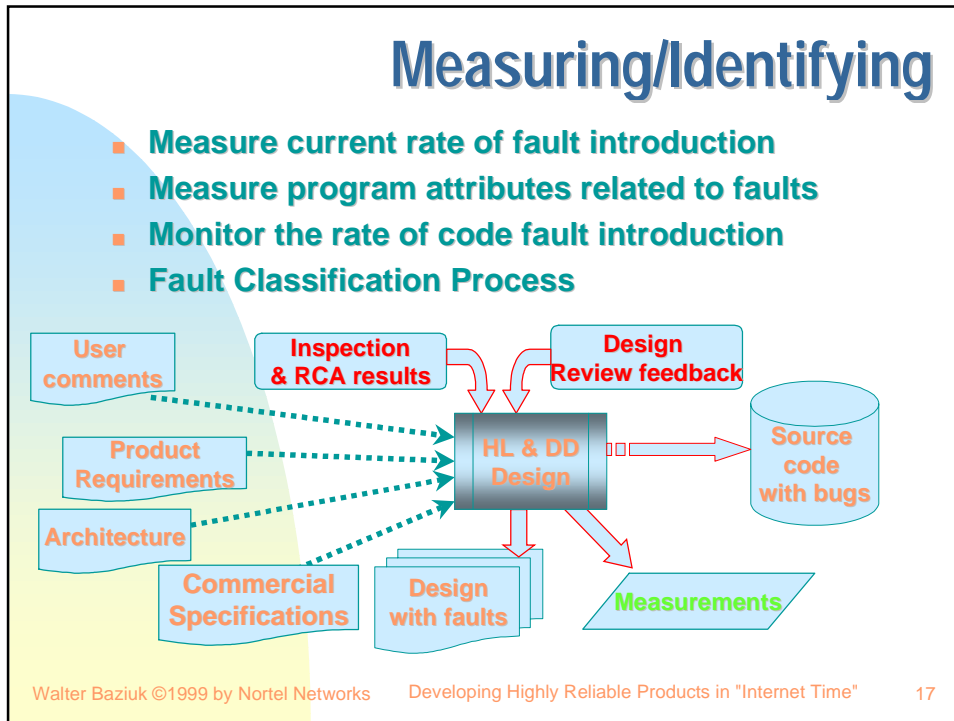
How many bulbs would have to be found and fixed before we could ship?



If we developed 100,000 LOC, ran 1000 test cases and 42 failed;

Good Question!

How many LOC would have to be identified and fixed before we could ship?



Agenda

- Identify key market/customer requirements
- Manage product reliability and quality
- Accelerate software development
 - ◆ How to work in small highly dynamic teams
 - ◆ How to break your product development cycle into shorter doable units.
 - ◆ How to work with the up-front and down stream staff to deliver a highly reliable product.

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 19

Crossfunctional Knowledge Sharing: For the Future

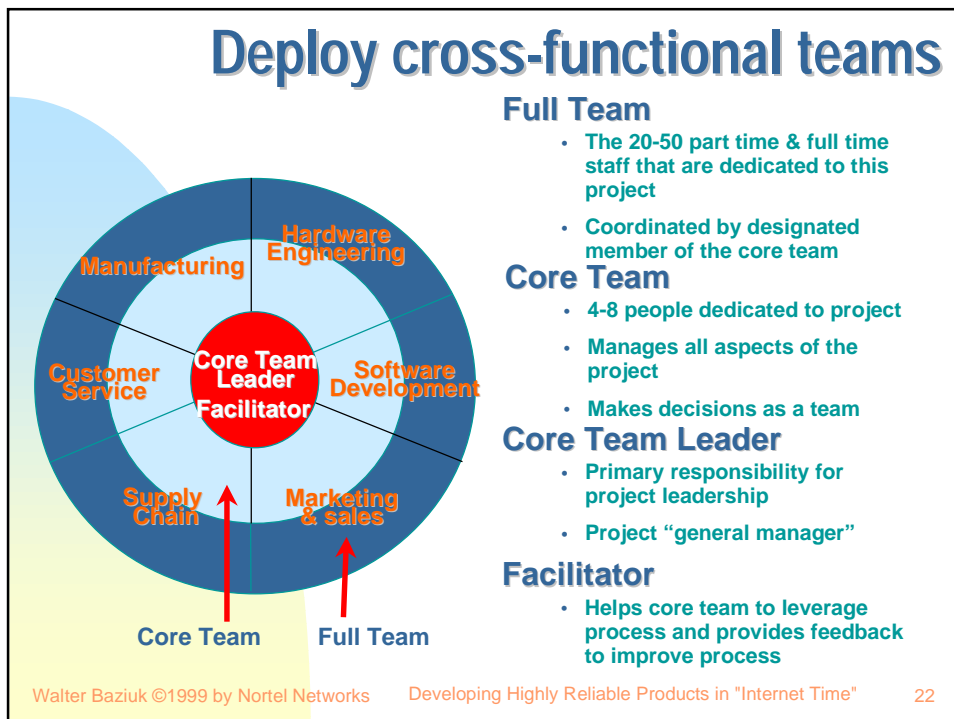
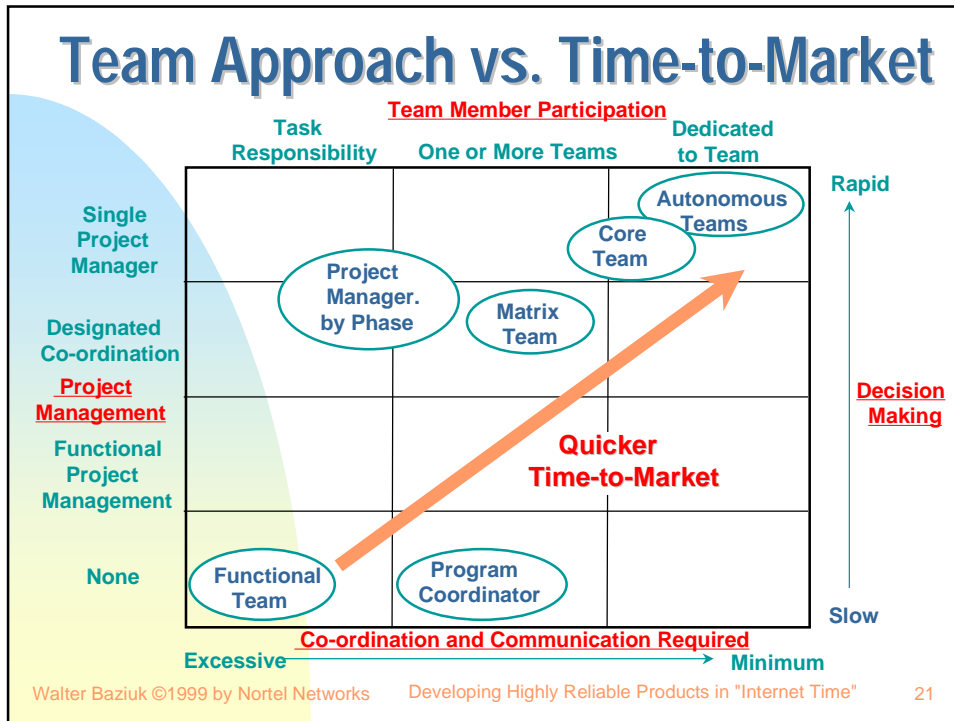
The diagram illustrates the intersection of four key functional areas: Project Management, Product Development, Business & Product Managers, and Quality & Process Engineering. These areas converge on a central point, with arrows indicating their impact on various business goals:

- Project Management** (top-left circle) is linked to:
 - Metrics
 - Measurements
 - Planning
- Product Development** (top-right circle) is linked to:
 - Features
 - Enhancements
 - Maintenance
- Business & Product Managers** (bottom circle) is linked to:
 - Cost
 - Marketing
 - Portfolio planning
- Quality & Process Engineering** (right side) is linked to:
 - Quality & Process Engineering
 - Small teams

Additional strategic goals mentioned on the left side include:


- Time to market enhancements
- Best practice deployment

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 20



Dedicated Teams

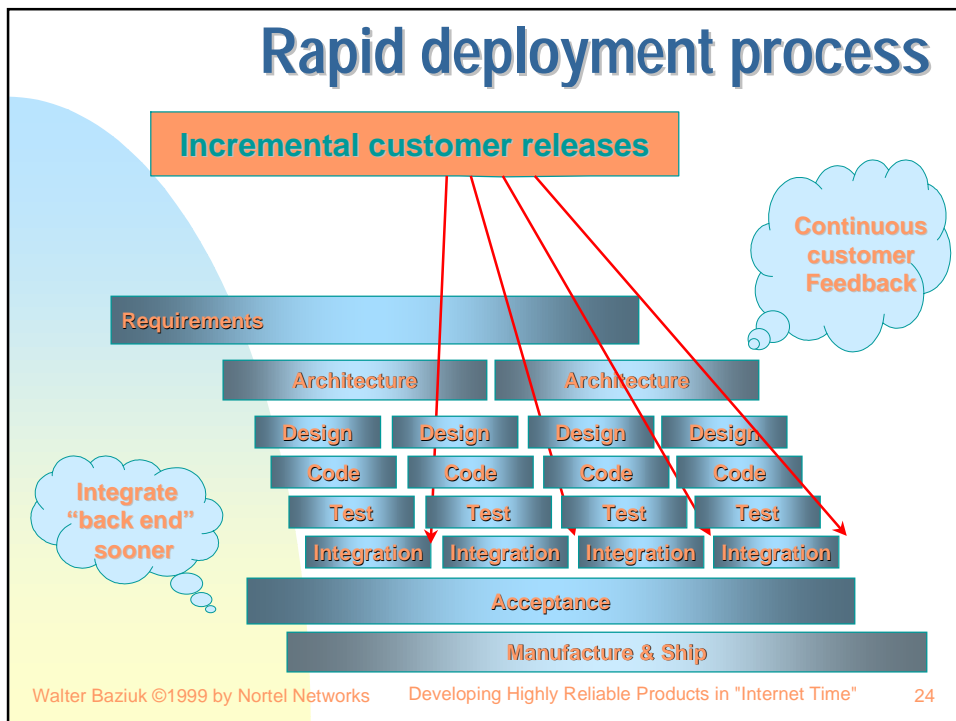
- Resource fragmentation is one of the greatest sources of poor productivity
- Long development cycles contribute to fragmentation
- Fragmentation is a cultural issue, often difficult to change
- The core team should be dedicated 100% of the time. Alarm bells should be sounded if any core team member's time starts to fall below **75%**.

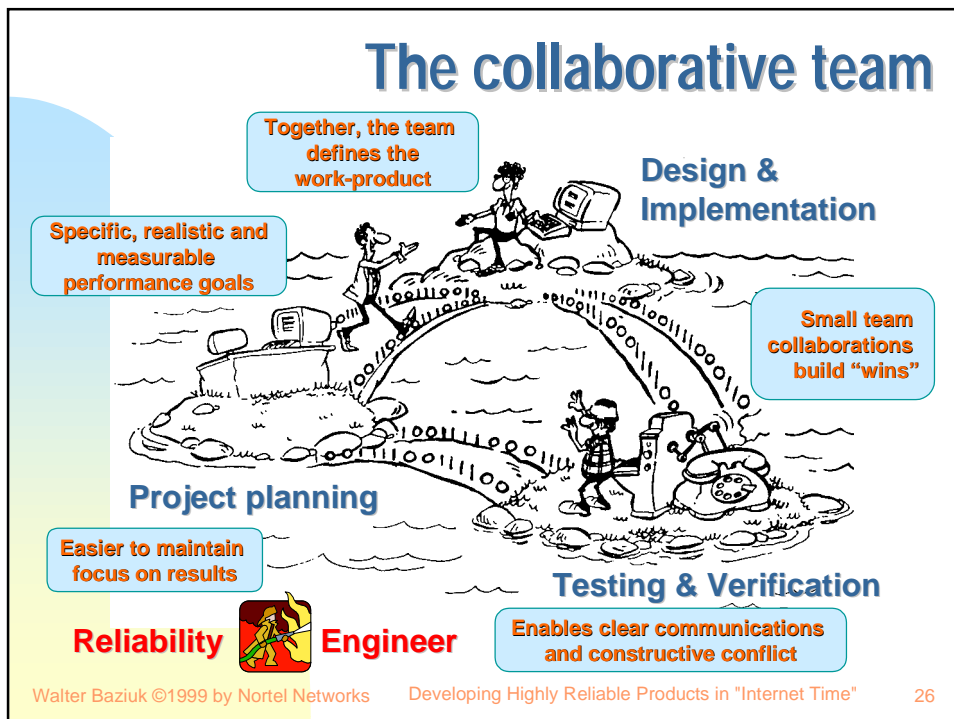
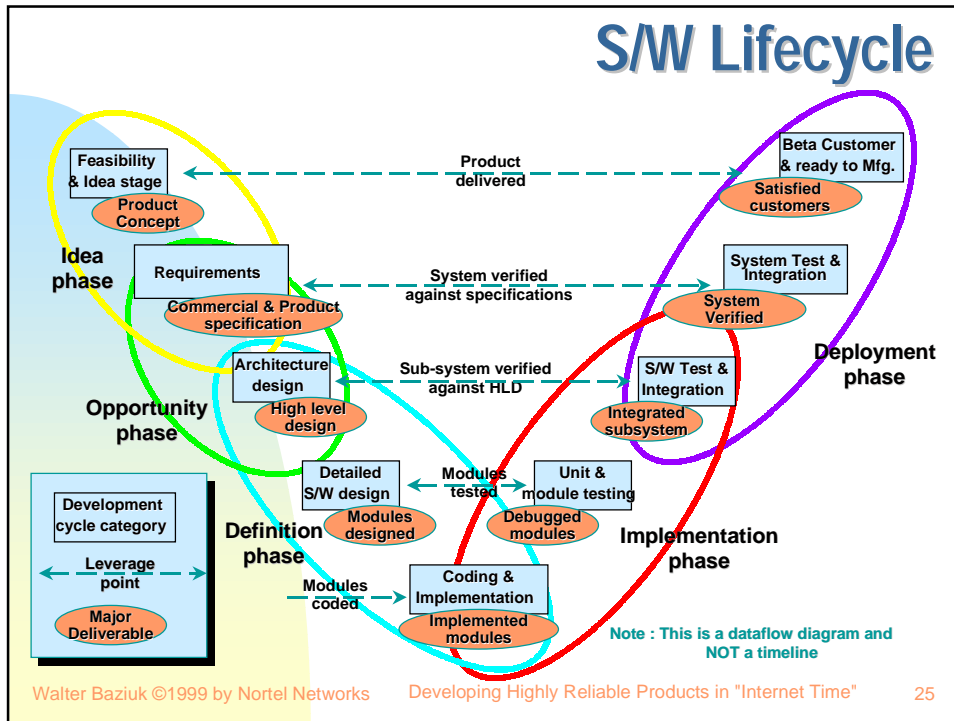


"A team is a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable."

Katzenback & Smith, The Wisdom of Teams

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 23





Recommendations

- **Once decisions are made, move forward!**
- **Reduce development projects to < 50 people**
- **Deploy cross-functional empowered teams**
- **Limit development time to < 6 months**
- **Define your objectives and confirm market opportunities each release**
- **Involve customer at each phase & release**
- **Deploy incremental releases to customer for feedback and approval**
- **Once decisions are made, move forward!**

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 27

Closing Remarks

Those who cannot remember the past are condemned to repeat it!
George Santayana, 1863 - 1952



Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 28

Developing Highly Reliable Products in "Internet Time"

Applications/
Solutions

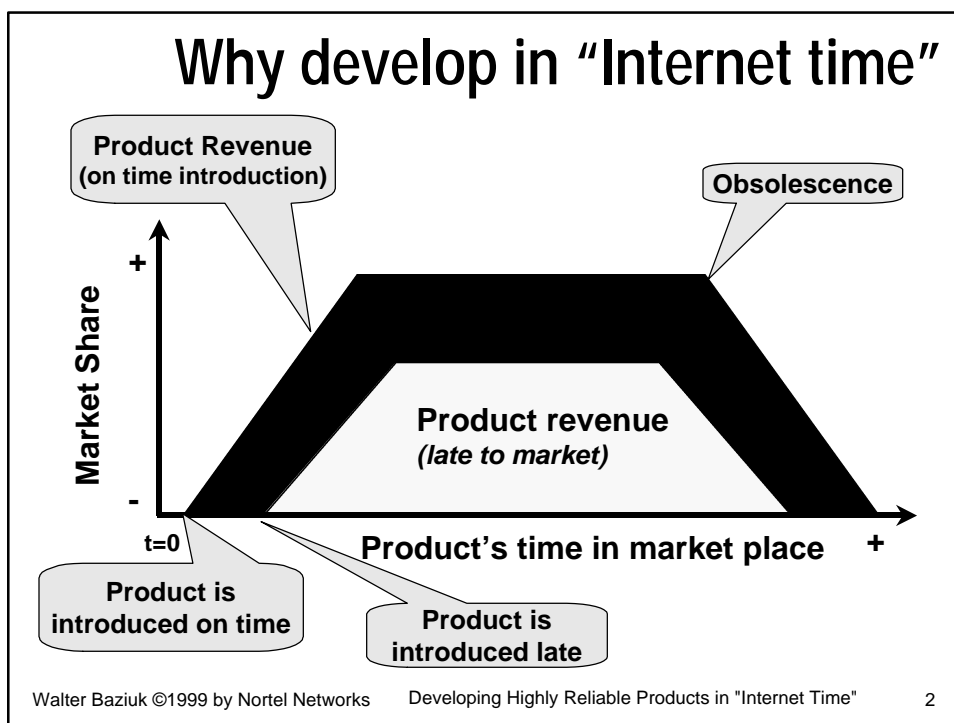


Track 5A

Walter Baziuk
baziuk@nortelnetworks.com



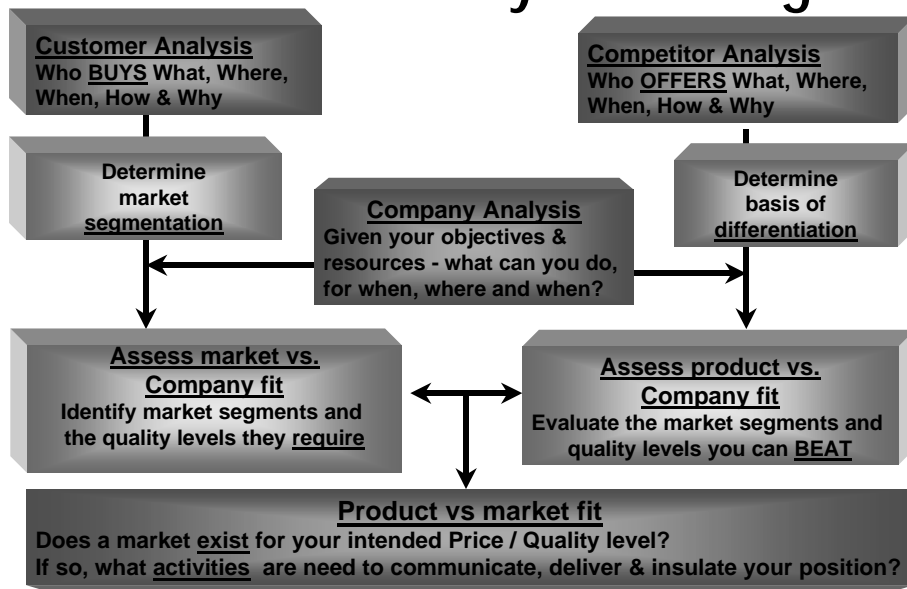
QWE99 - Brussels, Belgium - November 3, 1999

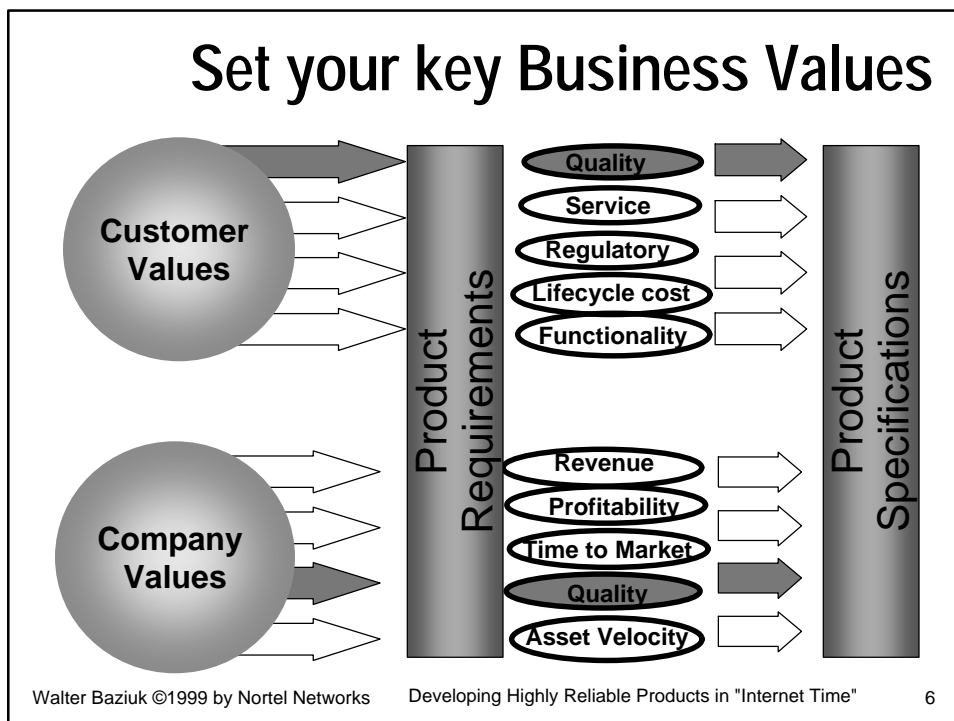
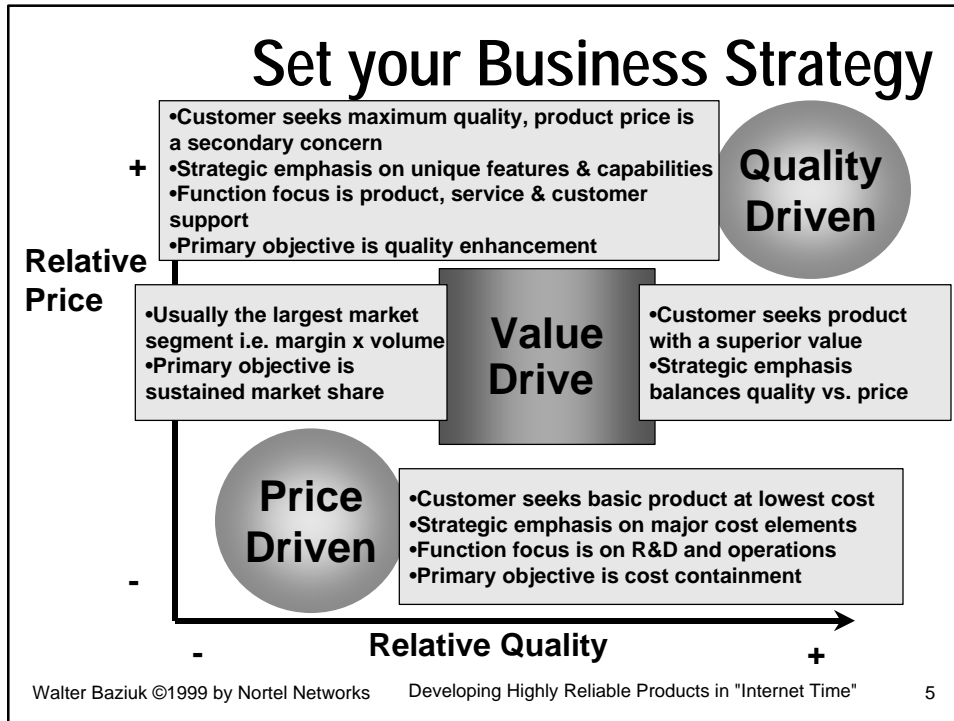


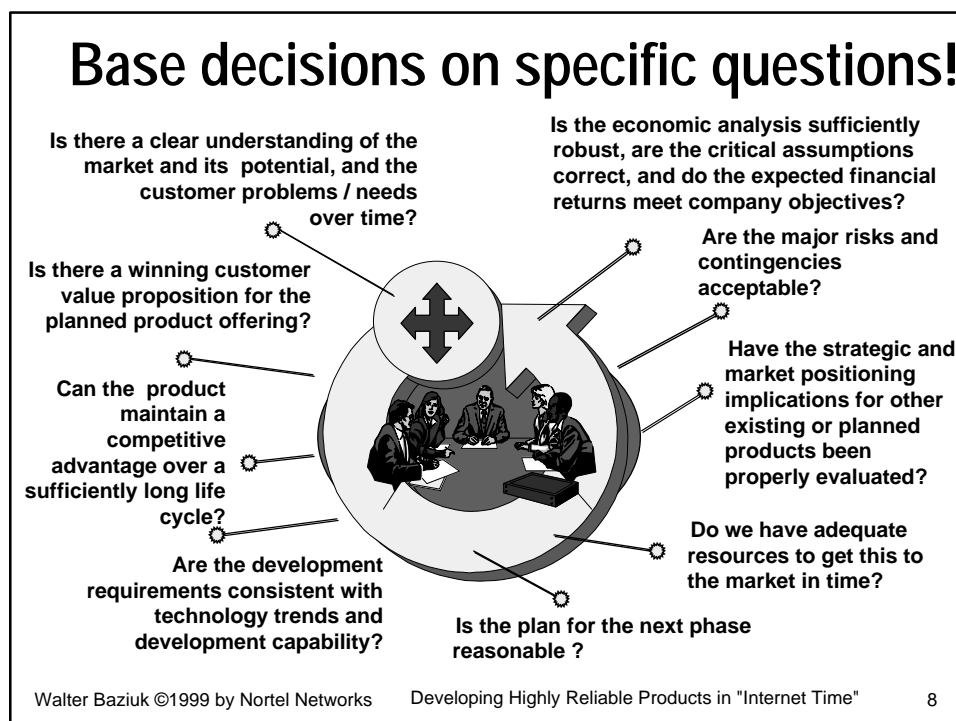
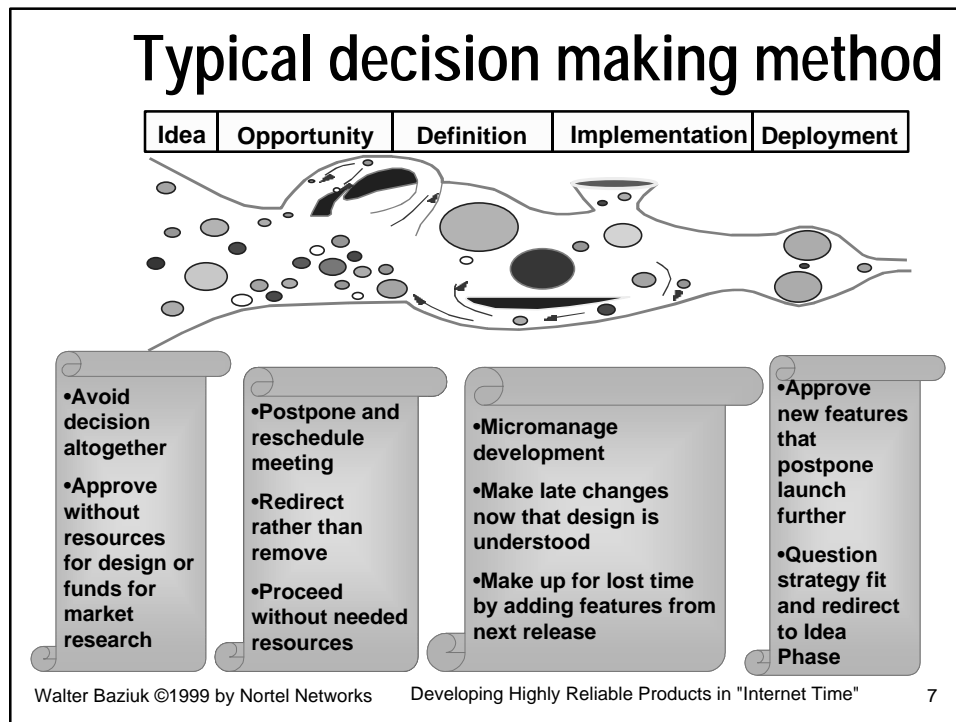
Agenda

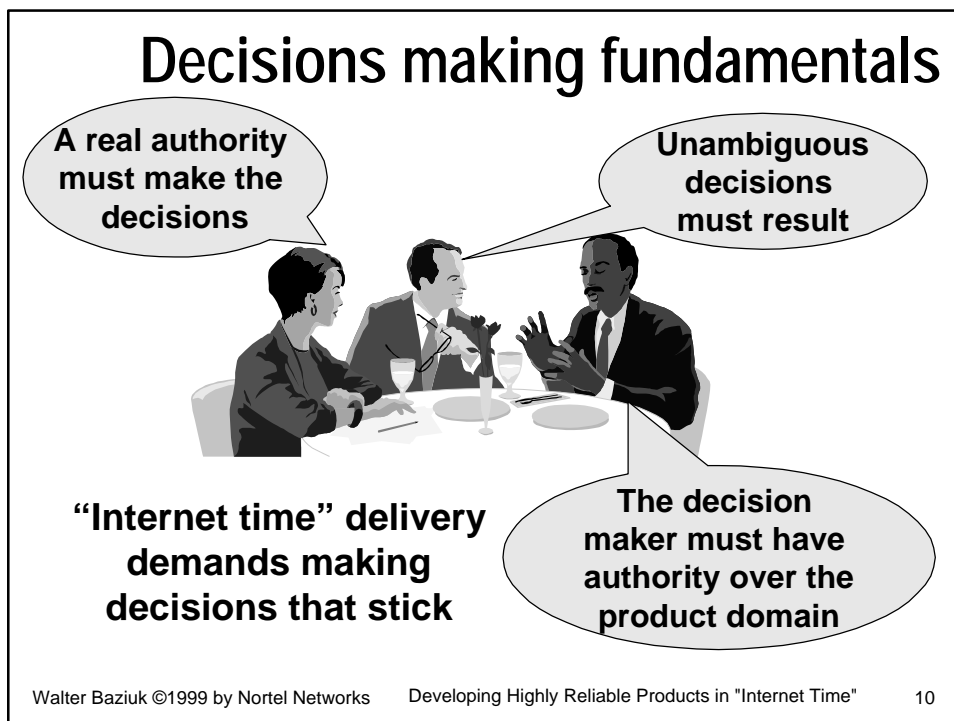
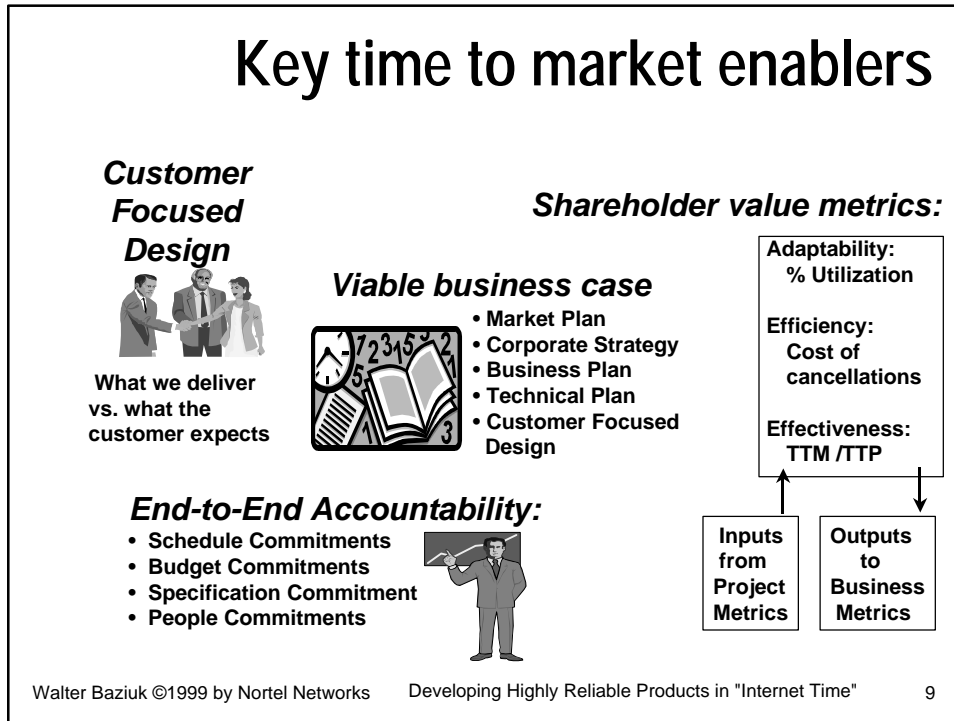
- ➔ **Identify key market/customer requirements**
 - ◆ Work with your customer determine "which features are in and which aren't" in each release.
 - ◆ Manage requirements churn and still deliver on time.
 - ◆ Get the customer involved earlier and have them witness intermediate results.
- **Manage product reliability and quality**
- **Accelerate software development**

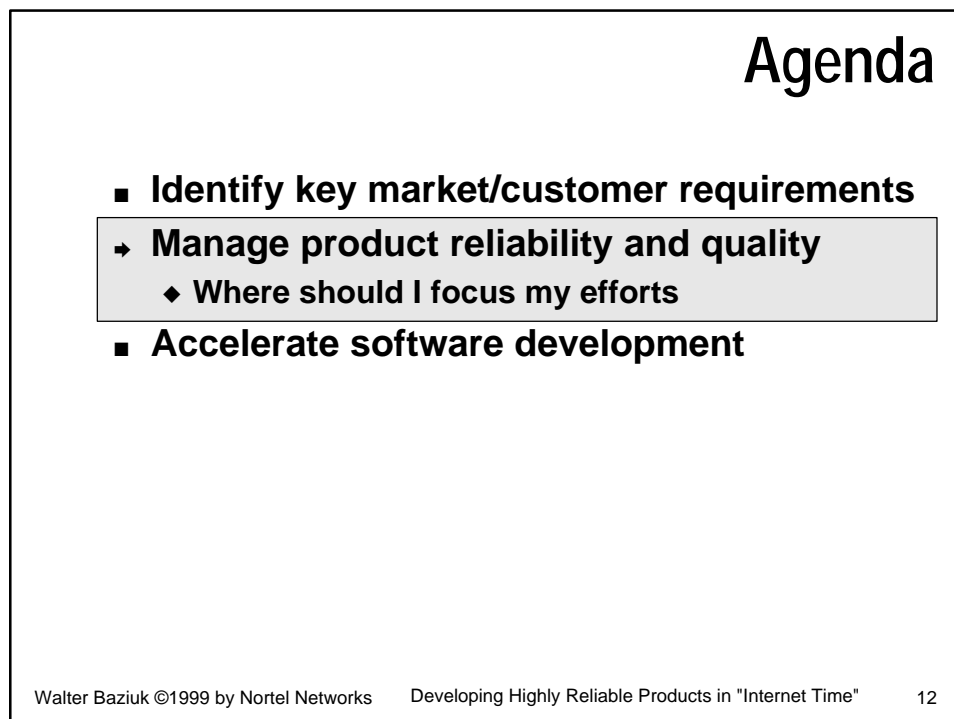
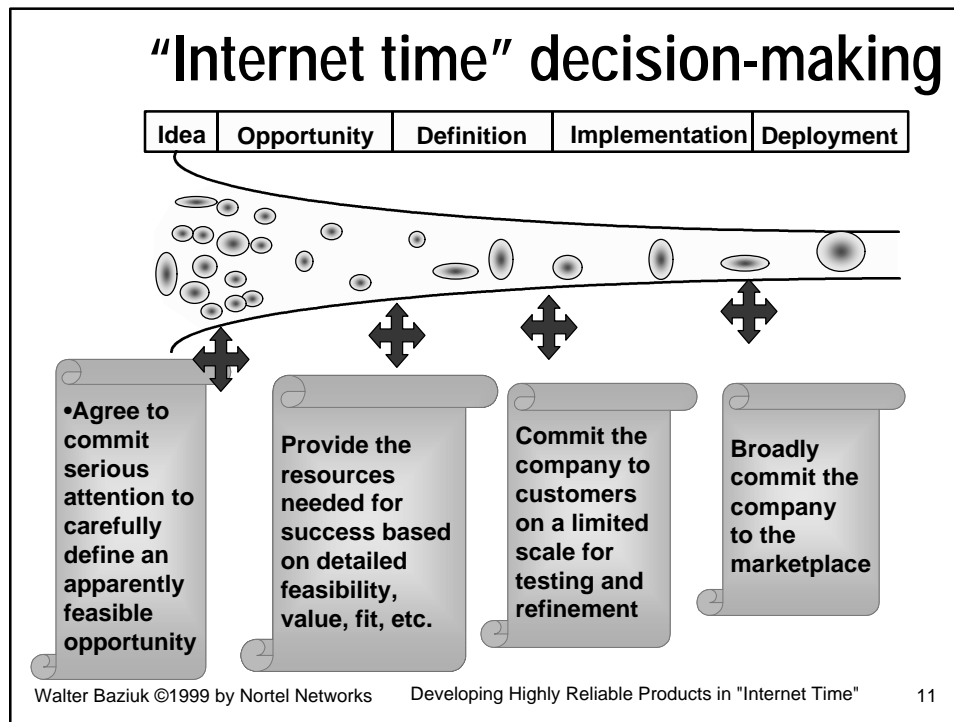
Define your strategic fit











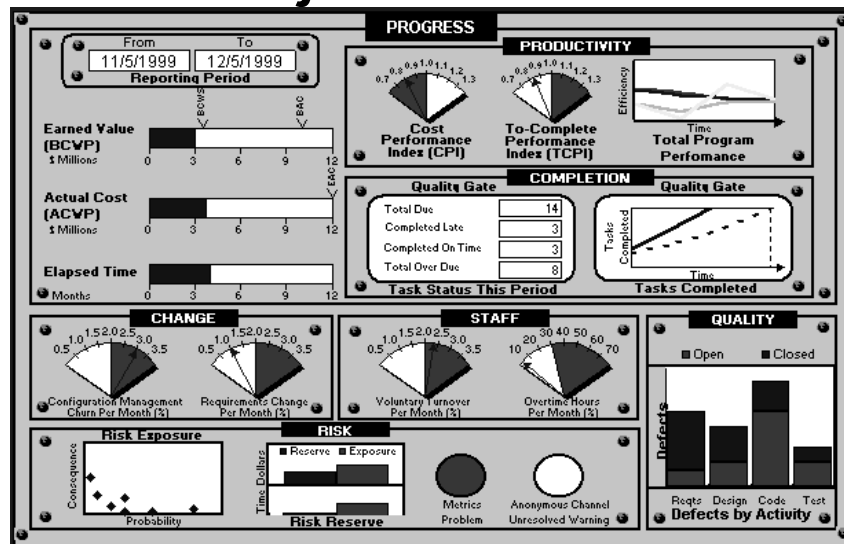
Competitive Intelligence study: recommendations

| Company | A | B | C | D | E | F |
|--|-----|-----|-----|-----|-----|----|
| Strategic Recommendations | | | | | | |
| Allocation of time in early stages | + | ++ | | | +++ | + |
| Project management Leadership | ++ | +++ | | | ++ | |
| Customer focus | +++ | ++ | +++ | | | |
| Internal communication of best practices | ++ | + | | +++ | | ++ |
| Compensation Package | +++ | + | ++ | +++ | + | |
| Tactical Recommendations | | | | | | |
| Build Concurrent Engineering teams at working level | +++ | ++ | +++ | | | |
| User Requirements - short distance between customers and designers | +++ | +++ | +++ | + | | ++ |
| Automated Test Tools | + | ++ | ++ | | | |
| Code reuse | ++ | ++ | | + | | |
| Limit team size | | +++ | +++ | | +++ | + |
| CASE tools | +++ | | | | | |

Company strengths:
 + mild
 ++ strong
 +++ very strong

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 13

Project office "dashboard"



From: Software Program manager network <http://www.spmn.com>

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 14

Fault Classification: trend vs. phase

| Product Phase | Activity / Source of Fault | Faults / KLOC | % contribution | Faults per phase | Cumulative Faults |
|---------------------|--------------------------------|---------------|----------------|------------------|-------------------|
| Requirements | Product Requirements Document | 2.5 | 5.0% | 5% | 5% |
| Definition | Product Objective Document | 3.5 | 7.0% | 24% | 29% |
| | Product Specification Document | 8.5 | 17.0% | | |
| Design | Architecture Inspection | 5.0 | 10.0% | 25% | 54% |
| | Design Inspection | 7.5 | 15.0% | | |
| Coding | Coding | 6.4 | 12.8% | 13% | 67% |
| Development testing | Module Test plan | 0.5 | 1.0% | 20% | 87% |
| | Module Test | 4.5 | 9.0% | | |
| | Unit Test plan | 0.5 | 1.0% | | |
| | Unit Test | 4.5 | 9.0% | | |
| Integration testing | Functional Test plan | 0.5 | 1.0% | 13% | 100% |
| | Functional Test | 3.5 | 7.0% | | |
| | System Test plan | 0.3 | 0.6% | | |
| | System Test | 2.0 | 4.0% | | |
| | Regression Test plan | 0.0 | 0.0% | | |
| | Regression Test | 0.2 | 0.4% | | |
| Post Delivery | Customer Problem reports | 0.1 | 0.2% | 13% | 100% |
| Total | | 50 | | | 100% |

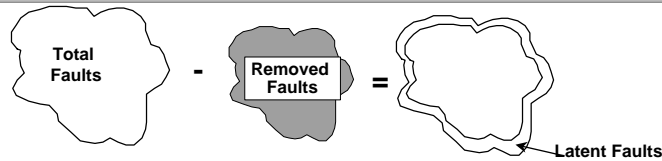
Faults/KLOC = (Major faults + (Minor faults)/ 3) / Total KLOC for product
 Major faults = Service affecting problem
 Minor faults = Non-service affecting problem

Latent Faults

If we manufactured 100,000 light bulbs,
sampled 1000 and found 42 to be defective;

4200

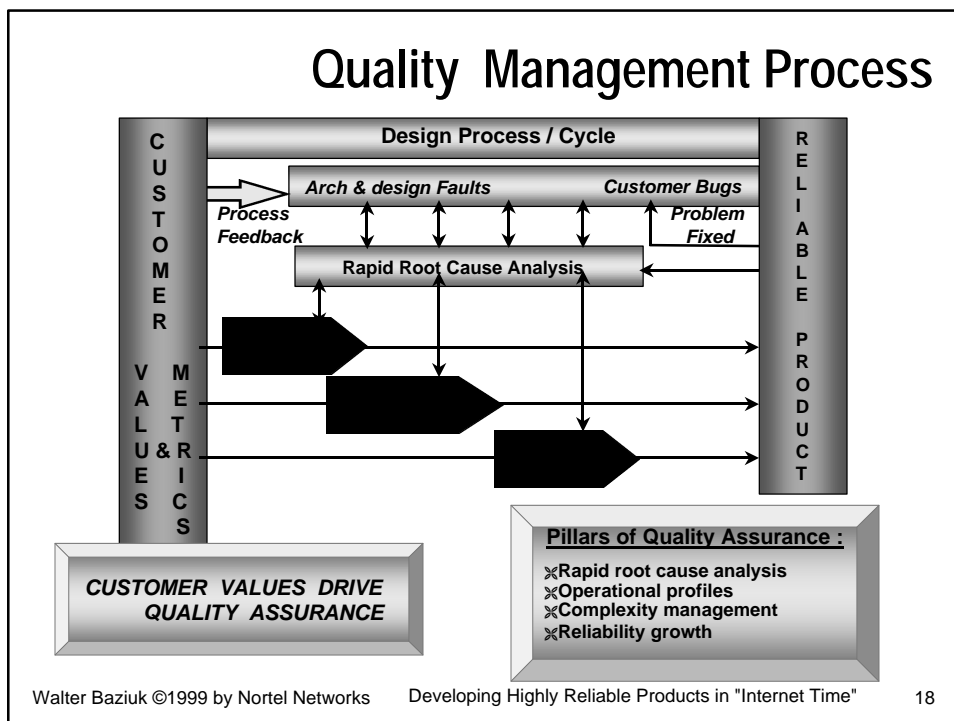
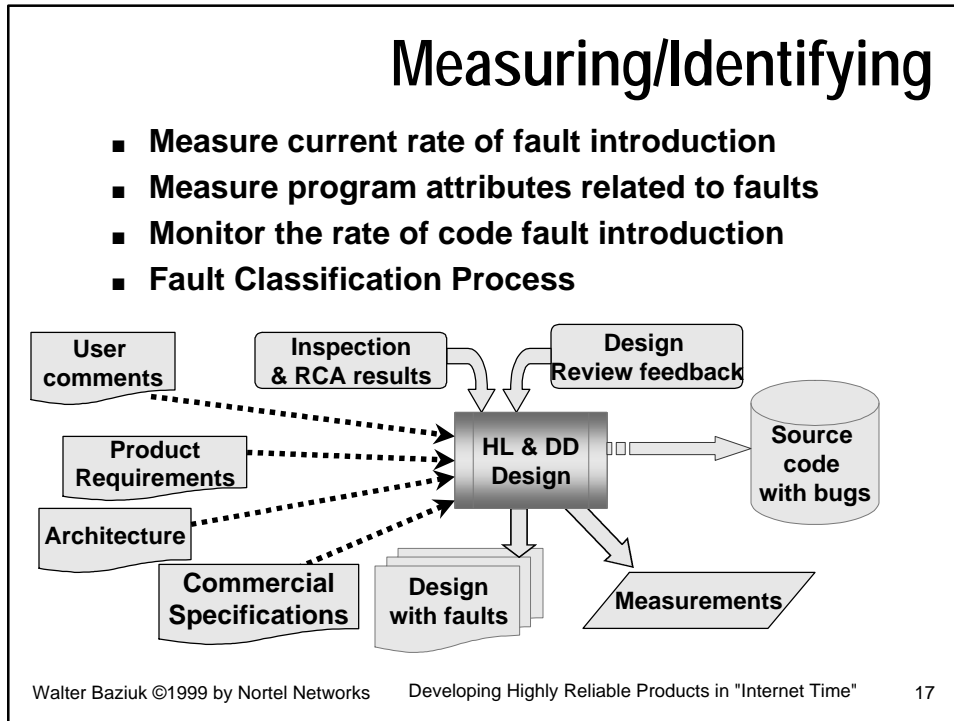
How many bulbs would have to be found and fixed before we could ship?



If we developed 100,000 LOC,
ran 1000 test cases and 42 failed;

Good Question!

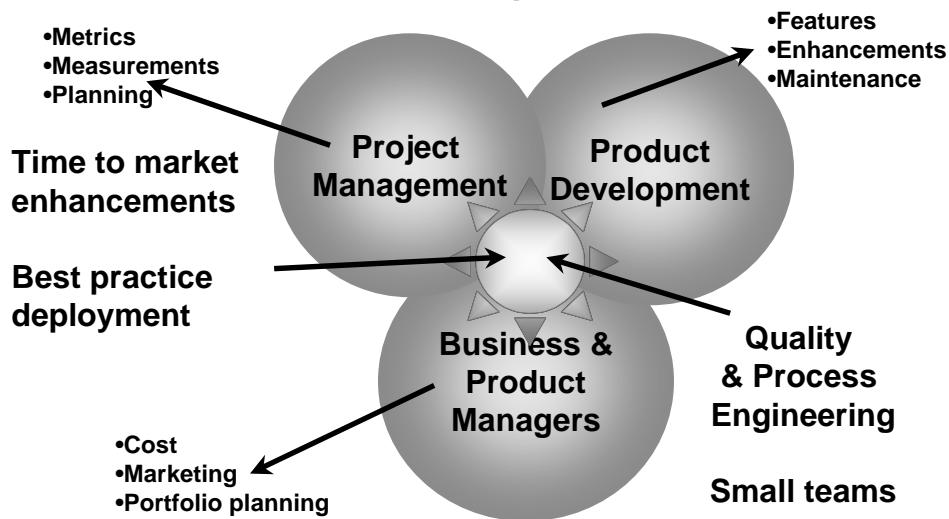
How many LOC would have to be identified and fixed before we could ship?

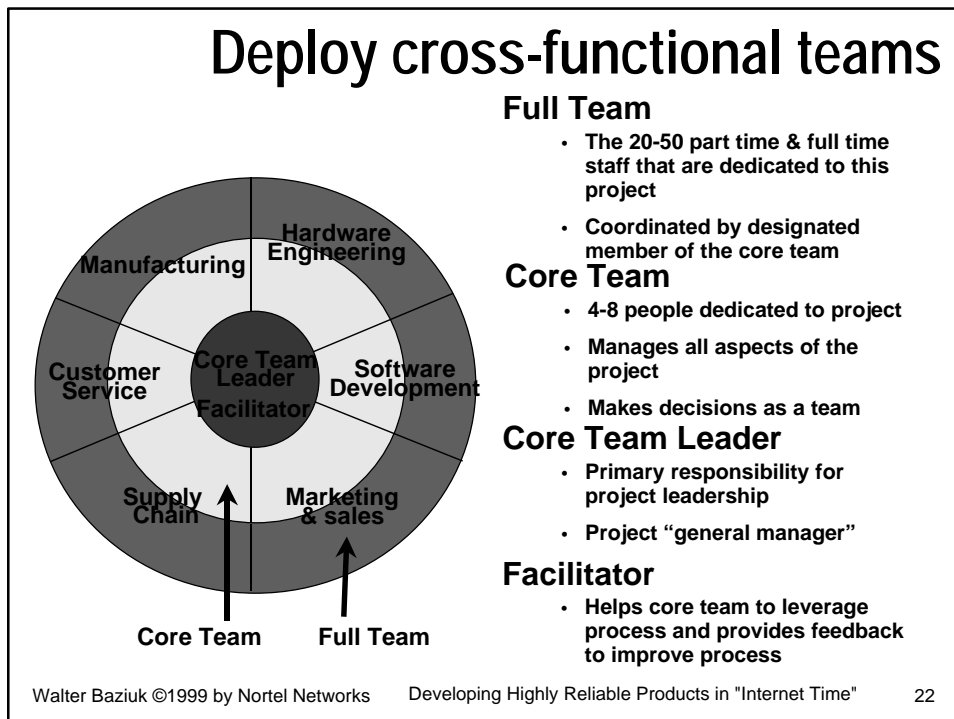
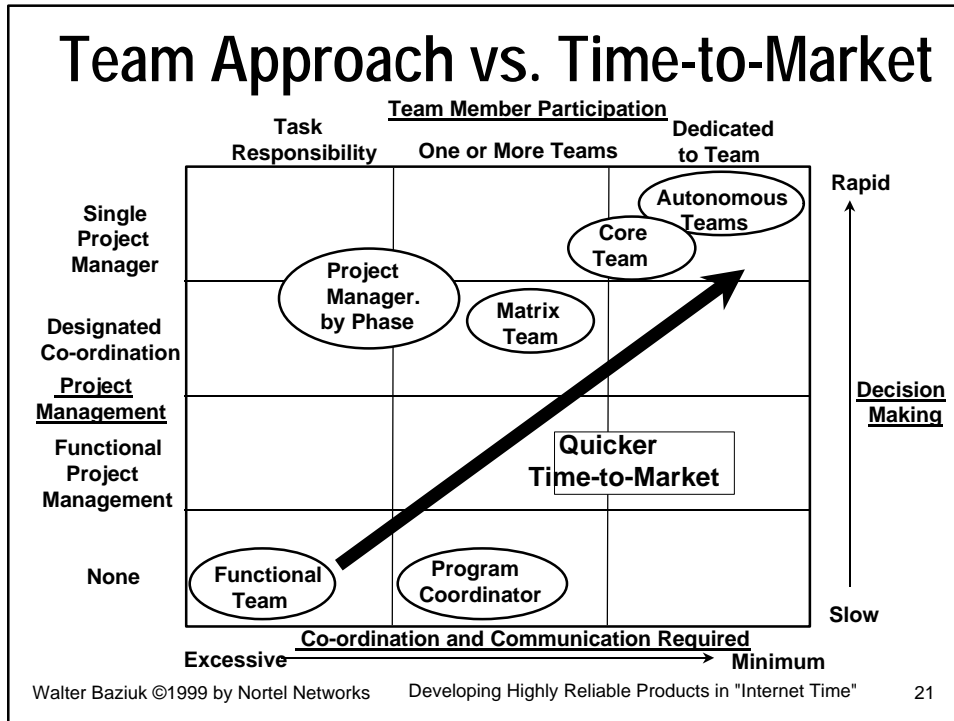


Agenda

- Identify key market/customer requirements
- Manage product reliability and quality
- ➔ Accelerate software development
 - ◆ How to work in small highly dynamic teams
 - ◆ How to break your product development cycle into shorter doable units.
 - ◆ How to work with the up-front and down stream staff to deliver a highly reliable product.

Crossfunctional Knowledge Sharing: For the Future





Dedicated Teams

- Resource fragmentation is one of the greatest sources of poor productivity
- Long development cycles contribute to fragmentation
- Fragmentation is a cultural issue, often difficult to change
- The core team should be dedicated 100% of the time. Alarm bells should be sounded if any core team member's time starts to fall below 75%.

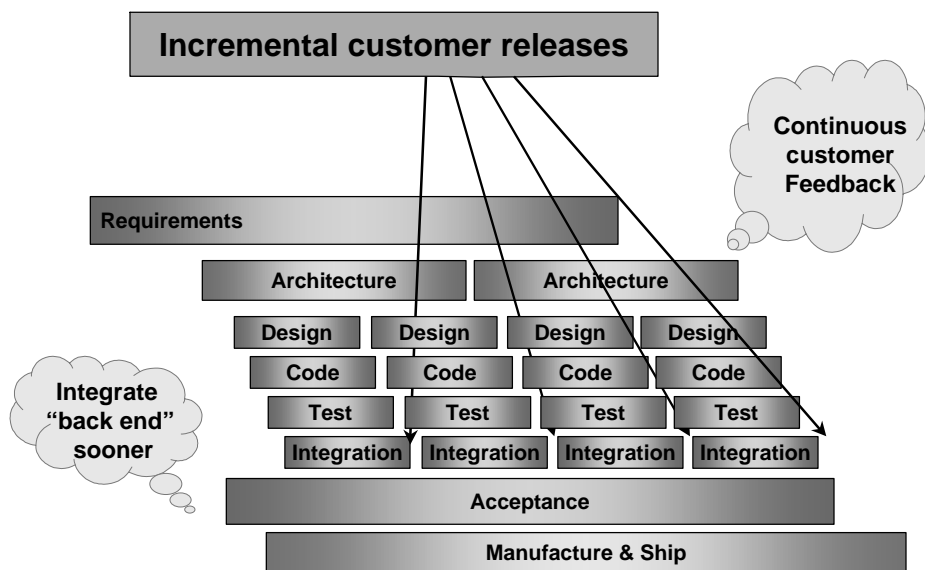


"A team is a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable."

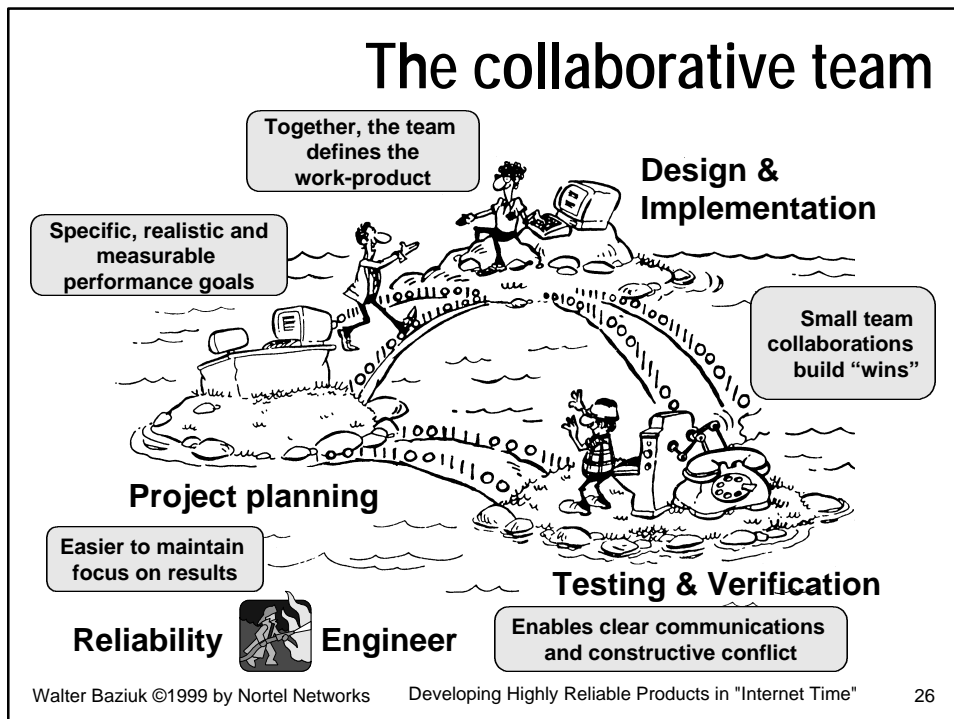
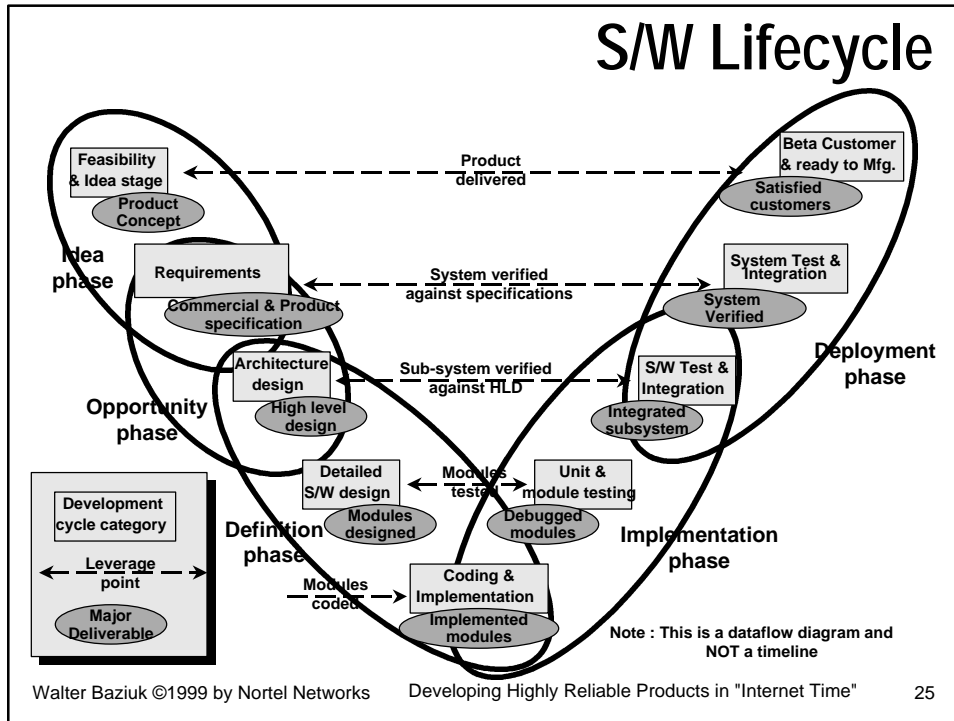
Katzenback & Smith, The Wisdom of Teams

Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 23

Rapid deployment process



Walter Baziuk ©1999 by Nortel Networks Developing Highly Reliable Products in "Internet Time" 24



Recommendations

- **Once decisions are made, move forward!**
- **Reduce development projects to < 50 people**
- **Deploy cross-functional empowered teams**
- **Limit development time to < 6 months**
- **Define your objectives and confirm market opportunities each release**
- **Involve customer at each phase & release**
- **Deploy incremental releases to customer for feedback and approval**
- **Once decisions are made, move forward!**

Closing Remarks

Those who cannot remember the past are condemned to repeat it!

George Santayana, 1863 - 1952



An Efficient Measurement for Assertion Placement

Jin-Cherng Lin & Szu-Wen Lin
 Dept. of Computer Science and Engineering
 Tatung University
 Taipei 10451, Taiwan, R.O.C
Jclin@cse.ttit.edu.tw
Swlin061@ms3.hinet.net
 Tel: +886-2-25925252 ext 3295

Abstract

Software testing is one of the most powerful methods to improve the software quality directly. Usually, testing costs on the large scales of 50% during software system development phase. How to reduce the testing cost thus becomes an important issue. Software Testability, the degree for software to reveal its faults during testing, is an important issue for quality assurance and verification. When testability is low, testers often want advice on how to increase it. We propose using data state errors measurement for assertion placement. Software assertions are one relatively simple trick for improving testability. When knowing where data state errors appear likely to hide from a test, we have insight into where assertions are beneficial. We explore using data state errors that measure testability as a method for where and to inject software assertions in their placement.
Keywords: software testability, assertion placement, software reliability, design for software testability

1. Introduction

Software reliability becomes significant while software applies to more critical applications. High reliability software means that the software works properly and meets its specification. How to promote the reliability of software is one of the most noticeable things for the software system developers. Software testing is one of the most powerful methods to check whether the software errors occur. It improves the software reliability directly [Yang].

Software testing has been proved to be an important and valuable activity for determining whether the software system has faults. The purpose of testing is to evaluate if a software system meets its specification and to evaluate the correctness, reliability, and failure rate of the software [Free91]. Software testing is performed

for two reasons: (1) to detect and then remove faults, and (2) to estimate the reliability of the code [Voas94]. Besides, testing costs on the large scales of 50% during software system development phase [Mos193, Boeh75]. How to reduce the testing cost thus becomes an important issue in the software engineering strategies.

Software assertion approach is one kind of software testing techniques. An assertion is a test on the state of an executing program or a test on some portion of the program states. Typically, software testing checks the correctness of values only after they are output. In contrast, assertions check intermediate values. A benefit of checking values internally is that we know as soon as possible whether the program has entered into an erroneous state [Voas95].

We assume that the assertion evaluates to TRUE when the corresponding internal state is satisfactory and FALSE otherwise. In this methodology, if an assertion evaluates to FALSE, then we consider the execution of the program to have resulted in failure, even if the eventual output is correct according to the specification [Voas95]. Hence we refine the conception of failure in the software testing: a failure occurs if the output results an unexpected incorrect state or an assertion fails.

Besides, the software designers should apply some kind of techniques to assist the software testing process in order to reveal errors easily and effectively. Software testability ought to be considered and emphasized during the design of software. The software testability is an ability to reveal errors while testing the software in various testing phases. To make a software "more testable" means the software system can easily reveal errors and can be checked easily and precisely against specifications. Software testability and software design for testability must get its proper attention in order to reduce testing cost and promote software reliability.

An assertion is a method for increasing the reliability of software by increasing the dimensionality of the output space. As long ago as the 1975 International Conference on Reliable Software, several authors described systems for deriving runtime consistency checks from simple assertions [Stuck75, Boeh75, Yau75]. Assertions have been recognized as an effective method for detection of software faults. Assertion features are available in many high-level formal specification languages, as well as some programming languages (such as Turing [HC88] and Eiffel [Mey88]). Such languages can be used to specify system behavior at the requirements and/or design levels [Rose92]. But those authors didn't mention to the three most important details: (1) where should we need to insert the assertions, (2) what kinds of assertions should be used, (3) and when assertions inserted, what is the result. We develop a technique called "state analysis" to arm the three details. This article brings up some proposal for the three details.

2. Related Works

2.1 Assertion

Assertions are used to check and maintain programs [Yang, Rose92, Biem95]. The primary goal in writing assertions is to specify what a system is supposed to do rather than how it is to do [Rose95]. When assertion tests fail, the program is aborted. The advantage of adding assertions is that we can make an automatic checking while developing the software system. The assertion method has been implemented in some languages such as Anna and Eiffel [Luck85]. Anna (Annotated Ada) uses comments to embed assertions; Eiffel uses object invariant inserted as pre-conditions and post-conditions to all operations on the object.

But there are still a lot of problems for assertions: (1) Previous assertion processing did not integrate easily with existing programming environments. (2) It is not well understood what kinds of assertions are most effective at detecting software faults [Rose95]. (3) The assertion techniques are little widespread used in practice. (4) Existed assertions processing system is not easy-to-use.

2.2 Annotation Pre-Processor for C

Annotation preprocessor (APP) for C [Rose95] uses assertion technique to check the consistency of software system. It expands the C extended comment to check the run time

consistency. It also provides some classification to prompt the user how and when to add the checking. But that the users have to learn new syntax makes its application limited.

2.3 Robust C

Robust C [Flat93] uses assertions to restrict and check the C programs semantics. In other words, Robust C enhances some insufficiency of C compiler such as out of range checking for arrays. The kind of enhanced C compilers have been developed by various versions of C or C++ languages. Such language can only solve limited errors because of C compiler's insufficiency.

For a lot of structure errors or errors about the specification of system output, Robust C can not check out them. At the same time, its preprocessor does not add checking at proper place in software. These are the disadvantages of Robust C.

2.4 Specification Based Test Oracle

Specification based test oracle [Rich92, Pete94] is designed from system specifications. It checks the outputs by corresponding with test data. The checking is said failed when the output does not match with the oracles. This kind of outputs checking, usually uses some particular specification languages, determines whether the system is correct or not. This method has to redesign different test oracle rules against different applications and thus can not reuse the test oracles.

2.5 Automated Test Oracles

Automated test oracles [Biem92] are used to check the outputs whether the system is correct or not. When a large number of inputs are executed, automated test oracle can help the user to check the outputs. However, it will take a few times for user to learn specification languages since such kind of methods usually bases on specification languages.

2.6 PIE Software Testability Model

Voas suggests that the software testability consists of three estimates P, I, and E [Voas92] at each location of the program:

Execution estimation E: The likelihood that the location is executed on inputs selected from the assumed input distribution of the software.

Infection estimation I: The likelihood that if a mutant exists at certain location, it will change the data state.

Propagation estimation P: The likelihood that if the

data state has been changed, the change will propagate to the output space.

The estimates of these three events occurred yields an estimate of the probability of failure that would occur when this location contains a fault. This is the testability of that location. By calculating the testability of each location in tested program, we can evaluate the software testability of a program in order to address the weak points of a software systems in the testing view. Besides, it can also be used to arrange the test schedule and test cost accurately.

3. Assertion Method in Software Programming

Assertions are used to test and maintain programs. Run-time assertion checking is a programming for validation “trick” which can help insure the program satisfies certain semantic constraints. Assertions are usually embedded into the programs to promote the ability of revealing faults. An advantage of using assertions is we can make an automatic checking [Yang, Rose92]. Assertions check the correctness of the program states. Typically, traditional software testing approaches only check the correctness of data states those are output. In contrast, assertions check “intermediate data values” those occur during the program is executing. Checking the intermediate data values benefits from that we know it immediately while the program has entered into an erroneous state.

Assertions have been recognized as an effective method for detection of software faults [Rose92, Biem95]. Assertion features are available in many high-level formal specification languages, as well as some programming languages (such as Turing [HC88] and Eiffel [Mey88]). But those authors didn’t mention the three key points: (1) where should we need to insert the assertions, (2) what kinds of assertions should be used, (3) and when assertions inserted, what is the result. This article proposes a method to solve these three key points.

In this article, we first consider what are the suitable locations to put the assertions. We advocate to put assertions only at some locations where the traditional software testing methods are unlikely to uncover faults. We use the measurement techniques and state analysis technique to locate the places where the faults maybe mostly hidden. Besides, we also consider what kinds of assertions should be put in order to

maximize effectiveness. Assertions are classified into many categories to study. The aim of classifying is to identify which category of assertions are suitable to be put in a specific location in the program to maximize effectiveness.

3.1 Terminology

- Anomaly: One of the most important concepts in software fault injection is the idea of anomaly. The anomaly is defined to some events that have the potential to alter software behavior through the corruption of some internal program state value. Anomalies occur at two levels: (1) internally in the program states, and (2) externally in the output space. From here on we refer to anomaly, we mean internal program state corruption [Gary97].
- Data state: The state of the program between one location and its succeeding locations is termed a data state. A data state includes the state of all defined variables, and fully determines where the control flow goes next.
- Fault: A fault is a defect in the program. To count as a fault, something must cause at least one of the input to result in failure.
- Failure: A failure is the event where a program’s output is not complied with the specification [BS96]. A failure occurs because the program enters an error state.
- Perturbation: Anomalies can get from any information source that feeds a program. Perturbation simulates the anomalies by replacing the original data with perturbed data.
- Perturbation function: The function specifies to perform data perturbation at locations in the program. Anomalies are simulated by the perturbation function.

3.2 State Analysis Estimate

State analysis measures the probability that the corrupted data state to transfer out. The state analysis produces a point estimate in $[0,1]$. We denote s_l as the estimate score of a source code at location l which represents a measure of how much impact the program state created by l has on the output computations of the program.

State analysis is a fault-injection technique that estimates the probability that an infected data state transfer to a location l will affect the succeeding locations. To make the estimate, state analysis repeatedly perturbs the selected variable. Hence state analysis is concerned with the

likelihood that the value in the following data state will be different than the value that is produced by the original location. State analysis is based on changes to the data states. To obtain the data states when executed to completion, we use a function based upon a random distribution termed a *perturbation function*. A *perturbation function* inputs a variable's value and produces a different value chosen according to the random distribution – the random distribution uses the original value as a parameter to produce the different value.

An algorithm for finding the state analysis estimate is:

1. Copy program P as P' .
2. Set **counts** to 0.
3. Select a variable or assignment x from the program P' . And, select one input data from the test case used for testing (the test data is P' 's input distribution domain D).
4. **Perturb** the sampled value of x of P' in the data state created after one location l , if l is executed and if x is defined. Else assign x a random value from the D , and execute the succeeding codes which uses x on both the perturbed data states (P') and original data states (P).
5. Execute the succeeding codes after location l , compare the outcome between the perturbed data states and the original data states. Increment **count** when they are different .
6. Repeat steps 3-5 n times.
7. Divide **counts** by n , get the estimate for each location.
8. Selected the minimum estimate from those using variable x locations.
9. Repeat steps 2-8, until each variable or assignment has been chosen.
10. When finished, we got some minimum estimate locations for all variables.

Assertions that are placed at each statement in a program can monitor the internal computations of a program execution. However, the advantage of inserted assertions everywhere in the program will come at a cost. The state analysis provides numerical scores of how likely faults are to be observable during testing.

The scores can rank ordered from highest (1.0) to the lowest (0.0). This ordering provides knowledge as to (1) which locations are likely to hide faults during testing, and (2) where assertions can be employed. If the state analysis estimate for location l is the lowest estimate score for some selected variable, then there is other locations using location l 's value may mask problem at l , and hence asserting on l allows us to be concerned with such problems.

To rank locations, we choice the estimate score which each selected variable perturbed data state and got the lowest score of estimate locations. By the way, we will got some locations will be the lowest score for each selected variable. And those locations are judged to be dangerously insensitive to faults. for a variable assigned a value at one of these dangerous locations. We will place an assertion at that location. The assertion is devised to reflect a required state of the computation of the location. This information must be extracted form the specification.

Then we will put assertions for all locations whose estimate score is lowest for the selected variable, and place those assertions in the locations.

The technique involves the testing characteristics of data and functions of the program into the source code to guard against the program faults. It promotes the software reliability after testing. Assertion checking is one such alternative, it is powerful, practical, scalable and simple to use. The goal of using assertions is to provide developers of large systems with applying assertions to their development efforts.

4. Classification of Assertions

Assertions have been recognized as a potentially powerful tool for runtime detection of software faults during debugging, testing and maintenance [DS92]. The assertions are often designed to check if the data values satisfy specified constraints. Assertions are defined in terms of specific data declarations, and they must be placed where the data values are referenced or modified.

In this chapter, we classify assertions for several types. Those types will check data invariant and pre- and post-conditions and others. We rank those assertions into different groups, one is block structure, which means that the program can see as some of the block unit modules into it. The block unit received argument from others block structure, not itself. It means that the argument is in other block unit output, but this block needs to use it.

The local statements are the sequence statements in the program, and its variable uses the value just from the sequence statement's output.

4.1 Block Structure:

The primary goal of specifying a block structure is to insure that the arguments, returned

value and states are valid with respect to the behavior of the block structure.

The common characteristic of all block structure constraints is that they are stated independently of any implementation for the block structure. That is, the block structure's behavior depends on the callers of the block structure. The block may be a unit module, function, subprogram, and so on. The constraints specify on pre-conditions and post-conditions.

4.1.1 Consistency:

Arguments in the block structure are often interdependent, for each block structure in the system, specify how every value of arguments depends on the values of other arguments.

We can use assertions to specify mutual consistency constraints and preconditions.

For instance, consider a language processing system that uses a routine called **getop** to get next operator or numeric operand in an input stream. As shown in below [Kern89], the routine takes the input stream and return value to main routine. The assumption checks that whether the syntax of the input is consistent with the value of argument. If the first input character is space or tab then skip it.

```

/* getop: get next operator or numeric
operand */
int getop(char s[] )
{
    int i,c;
    /* assert ( ( s[0] = c = getch( ) ) == ' '
|| c == '\t' ) */
    s[1] = '\0';
    if ( !isdigit(c) && c != '.' )
        return c;
    i = 0;
    if ( isdigit(c) )
        while ( isdigit( s[++i] = c =
getch( ) ));
    if ( c = '.' )
        while ( isdigit( s[++i] = c =
getch( ) ));
    s[i] = '\0';
    if ( c != EOF )
        ungetch( c );
    return NUMBER;
}

```

4.1.2 Dependency:

For each block structure in the system, it could use the post-condition to specify the returned value which depends on the argument values.

We can use assertions to state important aspects to specify the relationship between the returned value and its input entry.

For instance, as shown below, the post-conditions of the block illustrate this kind of assertion.

```

/* swap: interchange v[i] and v[j] */
void swap (char *v[ ], int i, int j )
{
    char *temp;
    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
    /* assert (v[i] && v[j] && v[i] != v[j]) */
}

```

4.1.3 State:

For each function in the system, specify whether argument values passed to function by references, or global variable value is to be changed or not by a function.

We can use assertions to specify how function changes the global state, and ensure that functions are called in certain contexts.

For instance, consider a language processing system that uses a routine call ***install** to put entries into hash table. As shown below, assume that hash table is searched using the subroutine **lookup**, which returns a non-zero pointer to a table entry if successful and null if unsuccessful. The assumption states that the argument to ***install** should have the entry in hash table. In particular, upon entry to ***install** a call to **lookup** with the same argument must return a non-zero or "true" result.

```

/* install: put ( name, defn ) in hash table */
struct nlist *install ( char *name, char *defn
)
{
    struct nlist *np;
    unsigned hashval;
    if ( ( np = lookup( name ) ) == NULL )
    {
        np = ( struct nlist * ) malloc(
sizeof( *np ) );
        if ( np == NULL || ( np -> name
= strdup( name ) ) == NULL )
            return NULL;
        hashval = hash( name );
        np -> next = hashtable[ hashval ];
        hashtable[ hashval ] = np;
    } else

```

```

        free( ( void * ) np -> defn );
    if ( ( np -> defn = strdup( defn ) ) ==
        NULL )
        return NULL;
    return np;
    /* assert ( np = lookup( name ) ) */
}

```

4.1.4 Membership:

For each block structure in the system, specify all constraints on the argument values and global states are the same type.

We can use assertions to specify the constraints and guard against the mishandling.

4.1.5 Non-Null Pointer:

For each block structure in the system, specify the pointer-valued arguments; returned values must not be null.

We can use assertions to specify when should pointers be non-null.

For instance, as shown in 4.1.2, the assumption “`assert (v[i] && v[j] && v[i] != v[j])`” specified on the `swap` shown in 4.1.2 illustrates an assertion that constrains a pointer argument to be non-null.

4.2 Local Statement

A program often contains a lot of sequences statements and complex control condition, which offer many opportunities for introducing faults. Assertions can be used to guard against such faults.

4.2.1 Complex Control Condition:

For each system, the implicit control condition of the default branch for if-else statement or switch statement is often stronger than other non-default conditions.

We can use assertions to specify the intended condition statements explicitly, and describe the limited domain.

For instance, as shown below, the sequence statements rather than taking an argument indicating the mid of elements it is got, instead makes that determination in its implementation.

```

/* binsearch: find x in v[0] <= v[1] <= ... <=
v[n-1] */
int binsearch(int x, int v[], intn)
{
    int low, high, mid;
    low = 0;

```

```

high = n - 1;
while (low <= high)
{
    mid = (low + high) / 2;
    if (x < v[mid])
        high = mid - 1;
    else if (x > v[mid])
        low = mid + 1;
    else /* found match */
        return mid;
    /* assert (low <= mid && mid <= high) */
}
return -1; /* no match */
}

```

4.2.2 Related Data Consistency:

It is often necessary to process related data in different ways in the system. We can use assertions to insure that the data remain consistent after processing. For instance, consider that inserts a new entry into a priority queue before performing other processing on the new entry. This might first use a loop to find where in the queue the new entry belongs. This might then use a separate check determining if the new entry was placed at the end of the queue, in which case the queue’s tail pointer would need to be updated. An assertion like the one shown below can be used to insure that the two parts of the insertion code treat the tail pointer consistently.

```

/* assert new_entry -> next !=0 ||
queue.tail == new_entry */

```

4.2.3 Intermediate State:

Assertion can monitor the intermediate state to check the dangerous locations. The effort of constructing the assertions is quick in automatic detection and isolation of faults that would have otherwise consumed as much as several hours of effort using more primitive debugging tools. And, it would be fruitful to stop and examine the systems with all the kinds of assertions that were most effective in uncovering faults.

The categories of assertions guard against many common kinds of faults and errors. Yet the very commonness of such faults demonstrates the need for an explicit, high-level, automatically checkable specification of required behavior.

5. Testability with Assertion

Are faults likely to hide after assertions are

added? We look at the situation with the code has no assertions, and then we will consider the situation with assertions are embedded.

For program \mathbf{P} without assertion in the test suite \mathbf{D} , we know the Testability $T(\mathbf{P})_D$ of the program \mathbf{P} . Now let \mathbf{P}_A represent program \mathbf{P} with assertions designed to boost the fault revealing ability of test suite \mathbf{D} . We denote as the Testability $T(\mathbf{P}_A)_D$.

From the Voas's sensitivity analysis, with the three analysis estimates, we will show the difference between program without assertions and program embedded with assertions.

Assertions increase the likelihood that the second and third conditions of the fault/failure model happen [Voas].

At first, we discuss the execution analysis estimate, $E(\mathbf{P})_D$ represents program \mathbf{P} without assertions in the test suite \mathbf{D} . And $E(\mathbf{P}_A)_D$ represents program \mathbf{P} with assertions in the test suite \mathbf{D} . We know that assertions were designed to boost the fault revealing ability of \mathbf{D} , assertions will not change the locations to execute or not. Hence, we can say that, $E(\mathbf{P}_A)_D$ is the same as $E(\mathbf{P})_D$. Then,

$$E(\mathbf{P})_D = E(\mathbf{P}_A)_D.$$

Second, for infection analysis estimate, $I(\mathbf{P})_D$ represents program \mathbf{P} without assertions in the test suite \mathbf{D} . And $I(\mathbf{P}_A)_D$ represents program \mathbf{P} with assertion in the test suite \mathbf{D} . From the sensitivity analysis definition, the infection analysis definition is the likelihood that if a mutant exits at certain location, it will change the data state. Assertions increase the likelihood of infection analysis of the fault/failure model. Then,

$$I(\mathbf{P})_D = I(\mathbf{P}_A)_D.$$

Third, for propagation analysis estimate, $P(\mathbf{P})_D$ represents program \mathbf{P} without assertions in the test suite \mathbf{D} . And $P(\mathbf{P}_A)_D$ represents program \mathbf{P} with assertion in the test suite \mathbf{D} . From the sensitivity analysis definition, the propagation analysis definition is the likelihood that if the data state has been changed, the change will propagate to the output space.

With the assertion characteristic, assertions will always propagate the situations to the output space. Then,

$$P(\mathbf{P})_D = P(\mathbf{P}_A)_D.$$

Hence, form the sensitivity analysis conditions we can get that:

$$E(\mathbf{P})_D = E(\mathbf{P}_A)_D.$$

$$I(\mathbf{P})_D = I(\mathbf{P}_A)_D.$$

$$P(\mathbf{P})_D = P(\mathbf{P}_A)_D.$$

And, testability is defined as execution * infection * propagation.

From those, we will conjecture that:

$$T(\mathbf{P})_D = T(\mathbf{P}_A)_D.$$

Our argument in support of this conjecture follows: if the assertions placed into program \mathbf{P} (with the test suite \mathbf{D}) are never with the input from \mathbf{D} , then $T(\mathbf{P})_D = T(\mathbf{P}_A)_D$; if one of the assertions is exercised by some input in \mathbf{D} , then it is possible that $T(\mathbf{P})_D = T(\mathbf{P}_A)_D$ will be true.

Since assertions directly affect propagation, then assertions appear to have an impact on fault detecting ability by increasing the output space.

For instance, as shown below, when the statement $a = a \text{ mod } 2$ has the lowest estimate score, what does it mean?

$$a = x + 1$$

.....

$$a = a \text{ mod } 2$$

We now denote the first a in the statement ($a = x + 1$) as a_1 , then a_2 in the right-hand side of the second statement ($a = a \text{ mod } 2$) and a_3 in the left-hand side. We use a_1 , a_2 , a_3 to substitute the variable a in the example.

When the variable a_1 changed by the variable x , the variable a_2 will use the value of a_1 . It means that if a_1 is incorrect than a_2 will be incorrect. Hence, the variable a_1 will execute by the example's statement. From the data flow viewpoint, a_1 's incorrect value will be right. But, the a_1 's incorrect value will change the a_2 's statement. What we think about a_1 and a_2 ?

From the a_1 's viewpoint, a_1 will changed by the variable x . For the statement a_1 below, we can say its execution will be correct, even though the variable x has error value. But the executing result will affect the a_2 's data state. And, for the single statement, we can not decide it's behavior is correct or with variable a . And, from Voas's testability definition, we discuss a_1 's execution didn't change. Because the statement execution will not affect by it's value. So we say the execution will not be affected.

Now we talk about the infection. If the variable x changed it's value, and it will transfer value to a_1 . The statement's infection estimate will not change. When the variable x in the right-hand side has an error state, it will still infect its state to a_1 . From the infection definition, the infection estimate will not affect anymore.

From a_2 's viewpoint, a_2 received a_1 's anomalies state and executed. At this time, the execution estimate will not be affected. When a_2 received a_1 's output as input, it will not be changed no matter the a_1 's state is correct or not. And the statement a_2 below, it will still be executed. Hence, execution estimate will not be affected. Now we

look inside the statement : the a_1 's anomalies will affect a_2 's input and a_2 's data state may not affect to a_3 . It means, the a_1 's anomalies will affect the statement. Then a_2 can't always reflect whether a_1 's data state is correct or not. Hence, the infection and propagation will be affected.

In other words, from the example, the statement with a_2 inside will have more probability of the low testability. So we can insert assertions to monitor its behavior. When the assertions inserted, the execution estimates will not be affected. Because of the assertions will not affect the statement's execution.

After the assertion has been inserted, for the infection and propagation estimate, the estimate score will be affected. The assertion can be used to monitor the statement behavior, using the statement's output as post-condition and the variable x 's input as pre-condition. With the monitor condition, when a_1 has error value and a_2 executed, the assertion can reflect the result. Therefore, the statement with assertion inserted will enhance the infection and propagation estimate score when the assertion reflecting its result.

Hence, the analysis method did not only monitor where is the variable state having anomalies, but also check where the anomalies will practically affect the program. By the way, we could improve the program's reliability and correctness.

6. Concluding Remarks

6.1 Conclusion

We have provided a methodology where we should insert assertions into the program, and our methodology predicting where faults will hide during testing. Assertions are applicable to software application, however the more critical the application is if an application only needs demonstrate modest degrees of quality, then the cost of assertions may not be warranted, as that quality can be demonstrated via basic types of testing.

Also note that there are different forms of assertions for different applications. Thus for specialized applications, specialized assertions may be necessary. Assertions are not the only verification and validation tricks that could be employed once it is known where testing is unlikely to be capable of detecting faults. Manual inspections, extensive unit testing, or formal analysis could also be applied to insure that defects

are not hiding.

Assertions are beneficial to software testing as "among the most significant ideas by testing and analysis researchers" [Oster92]. From our previous work where to insert assertions into the program, we believe that we have provided insights into why assertions work well and how to insert assertions.

6.2 Future Work

This article proposes the strategy and a technique of using assertions for software testing in order to reduce the testing effort and improve the software quality. The technique involves the testing characteristics of data and functions of the program into the source code to guard against the program faults. It promotes the software reliability after testing. Assertion checking is one such alternative, it is powerful, practical, scalable and simple to use. The goal of using assertions is to provide developers of large systems with applying assertions to their development efforts.

The future work includes (1) Expanding the assertion type to be more complete. (2) Use the strategy of assertion placement to guide how to insert suitable assertions in test steps. (3) Use the strategy and technique of assertion placement for software testing proposed in this article to develop a software tool to help the software design/test engineers in practical cases. (4) Build the tool to help the programmers and testers for checking the system.

References

- [Biem92]J.M. Bieman and H. Yin, "Designing for Software Testability Using Automated Oracles", IEEE International Test Conference, 1992, pp.900-907
- [Biem95]J.M. Bieman and H. Yin, "A Practical Approach to Programming With Assertions", IEEE Transactions On Software Engineering, Vol. 21, No. 1, January 1995.
- [Boeh75]B.W. Boehm, The High Cost of Software, in Practical Strategies for Developing Large Software System, Edited by E. Horowitz, Reading, MA: Addison-Wesley, 1975.
- [BS96] Antonia Bertolino, Lorenzo Strigini, "Using Testability Measures for Dependability Assessment", IEEE Transactions of Software Engineering, Vol. 22, number 2, Feb 1996.
- [Flat93] D. W. Flater and Y. Yesha and E. K.

- Park, "Extensions to the C Programming Language for Enhanced Fault Detection", *Software-Practice*, June, 1993, pp.617-628.
- [Free91] R. S. Freedman, "Testability of Software Components", *IEEE Transactions on Software Engineering*, June, 1991, pp.553-564.
- [Gray97] J.M. Voas and Gary McGraw, "Software Fault Injection", Wiley Computer Publishing, 1997.
- [HC88] Richard C. Holt and James R. Cordy. The Turing programming language. *Communications of the ACM*, 31(12):1410-1423, December 1988.
- [Kren89] Kernighan, Brian W., Ritchie, Dennis M., *The C programming language*. (2nd edition), New Jersey, Prentice Hall, 1989.
- [Luck85] D. Luckham and F. von Henke. An overview of ANNA, a specification language for Ada. *IEEE Software*, page 9-22, March 1985.
- [Mey88] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 1988.
- [Mos193] D.J. Mosley, "The Handbook of MIS Application Software Testing", Yourdon Press Computing Series, Englewood Cliffs, N.J., 1993.
- [Pete94] D. Peters, "Generating a Test Oracle from Program Documentation", *International Symposium on Software Testing and Analysis, SIGSOFT Software Engineering Notes*, Seattle, W.A., 17-19 Aug. 1994, pp.58-65.
- [Oster92] L. Osterweil and L. Clarke. A Proposed Testing and Analysis Research Initiative. *IEEE Software*, pages 89-96, September 1992.
- [Rich92] D.J. Richardson, "Specification-based Test Oracles for Reactive Systems", *ACM*, 1992, pp.105-118.
- [Rose95] D.S. Rosenblum, "A Practical Approach to Programming With Assertions", *IEEE Trans. on Software Engineering*, January 1995, pp.19-31.
- [Rose92] D.S. Rosenblum, "Towards a Method of Programming with Assertions", *Proceedings, 14th International Conference on Software Engineering*, Melbourne, Australia, 1992, pp.92-104.
- [Stuck75] L.G. Stucki and G. L. Foshee, "New assertion concepts for self-metric software validation," in *Proc. Int. Conf. Reliable Software*, ACM and IEEE Computer Society, Apr. 1975, pp. 59-71.
- [Voas95] J.M. Voas and K.W. Miller, "Software Testability The New Verification", *IEEE Software Journal*, MAY 1995, pp.17-28
- [Voas94] J.M. Voas and K.W. Miller, "Putting Assertions in Their Place", *Proc. of the International Symposium on Software Reliability Engineering*, Monterey, CA, November 1994. IEEE Computer Society Press.
- [Voas91] J. Voas, L. Morell, and K. Miller, "Predicting Where Faults Can Hide from Testing", *IEEE Software Journal*, March 1991, pp.41-48
- [Voas] J. Voas, L.Kassab, "Using Assertions to Make Software More Testable", To appear in *Software Quality Professional*.
- [Yang] S.C. Yang, "An Experimental Partial Oracle for Software Testability at Programming Step".
- [Yau75] S. S. Yau and R. C. Cheung, "Design of self-checking software," in *Proc. Int. Conf. Reliable Software*, ACM and IEEE Computer Society, Apr. 1975, pp. 105-113.

"Integrated Test Automation of IVR-Telephony Applications and Client-Server Call Center Applications"

Torsten Baumann; Test Team Leader,
Interactive Media Group,
Toronto, Ontario, Canada

Rex Black; President and Principal Consultant,
RBCS, San Antonio, TX

Serban Teodorescu; President and Principal Consultant,
STCS, Toronto, Ontario

Gordon Page; Consultant
RBCS, San Antonio, TX

Key Words: IVR, telephony, CTI, Client-Server, Test Automation,
Test System Fake Pipe (TSFP), integration, simulated calls,
Results, system under test.

Abstract

The Testing performed by our Test Group is Black Box Testing. This paper will discuss the following areas of our test effort in more detail:

- Client-server GUI testing
- CTI client-server testing
- IVR applications testing using simulated calls
- Product-wide integration testing using multi-tools (CallSim, Fake-Pipe, QA Partner) Approach
- Management considerations

Introduction

While test tools do exist for IVR Telephony, Client-Server and GUI Testing independently, there is not one that can test across the various systems. We have developed an approach that can be used in an integrated test environment.

The system under test (SUT) is described as follows. There are four main components:

- The Customer service application
- The IVR Telephony application
- The Content monitoring application
- The CTI server and PBX.

[See figure 1.0]

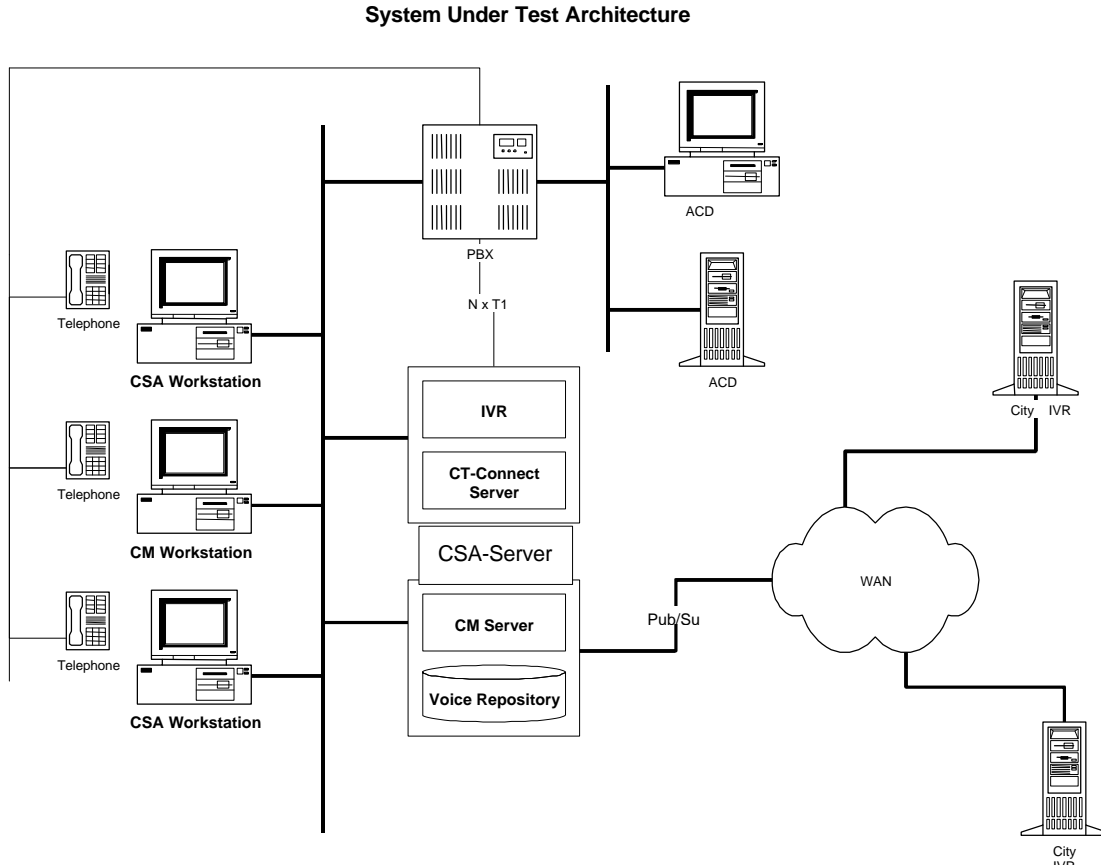


Figure 1.0

The Client-Server GUI Testing will discuss the obstacles involved in developing and executing tests on a multi-form application developed in VB for applications.

The CTI Client-Server Testing will discuss the successes and difficulties involved in testing a CTI application developed in Visual C++.

The IVR applications module will allow the audience to understand the complexities involved in developing a call simulator in Perl as well as Tcl for stress, performance and path coverage testing.

The Product-wide Integration Testing using multi-tools approach will focus on the successes and issues associated with developing an integrated automation suite for this project. We will describe how test tools and programming languages widely available on the market can be combined by means of a “Test System Fake Pipe” (TSFP) to effectively test the entire data flow of the integrated system under realistic stress conditions. All failures are captured in a single easy to read report generated after test execution. All control points between the systems under test will be held in the TSFP file.

Last but not least the paper will discuss the **management considerations** involved in leading a test team made up of consultants, full-time test engineers and student test technicians in a project as large and as complex as this one.

Who we are Software Testing, QA or QC?

Quality Assurance:

These teams *assure quality*. You cannot do that by testing alone. True QA Staff is extremely senior otherwise there will be an issue of credibility. If this is the group that assures quality then the rest of the company does not assure quality. A true QA group must be involved at every stage of development; it must set standards, introduce review procedures, and educate people into better ways to design and develop products. It helps a company prevent defects. Here at iMG management holds this role.

This is not us.

Quality Control:

They do a lot of inspection and have the power ensure that a defective product not be released or be removed from production. We recommend to management that a defective product be pulled from the hopper. The QC Group will provide management with the following:

- a) Reports,
- b) Information gathering,
- c) Software testing of various products.

This group is a management assistant in that it informs management of product problems and the severity. Ultimately management decides.

This is what we are called.

Software Testers:

This is really what we are.

Project at a Glance

This project as all projects has had its ups and downs. The overall approach that was used for the entire project was as follows:

- Developers would Unit/Component and String Test their code.
- Developers would install the SUT in the test environment and Test would then perform a smoke test to determine if the system that was installed is in fact testable.
- If not, development would roll back the test environment to previous build. If testable, the testing cycle would begin.

Component, Integration, and System Test

When we first arrived at iMG, no formal testing by a test organization occurred prior to release to a test city for acceptance test. This strategy proved to be risky, since the deployment of an untested release candidate to a test city could result in some amount of downtime for that city's customer service staff due to software defects thus having a revenue impact for the company.

We then proceeded in implementing a test team, separate from the development team, to perform formal testing against release candidates prior to acceptance testing. This testing was generally divided into three overlapping phases:

Unit/Component Testing would basically cover the following areas:

- States, Transactions, Code Coverage, Functionality, Interface.

System Test would focus on each individual system with stubs around the integrated pieces so as to verify if the CT application would work standalone or the CSA application would work standalone.

This in no way showed that CSA would communicate effectively with CT application or vice-versa. The following quality risks are checked for:

- Functionality, User Interface, States, Transactions, Data Quality, Operations, Stress/Capacity, Load, Error Handling/Recovery, Installation, Standards, Configuration Options.

Integration Test would focus on having all major components of the entire project being put together, so to say, in the test environment which is a replicate of the production environment so that it can be tested from a black box approach. The quality risks verified for here were:

- Component Interfaces, Capacity and Volume, Error Handling and Recovery, Data Quality, Functionality, Usability, Performance.

In terms of release management during testing, the common practice that we used is the use the concept of “cycles”. Each cycle takes place against a fresh build of the product, incorporating corrections that address all the “must-fix” bugs identified in the previous cycle.

A typical test phase, from the point of view of cycles, might follow the outline in the following table.

| Test Phase | Cycle One | Cycle Two | Cycle Three | Cycle Four | Cycle Five |
|--------------------|-----------|-----------|-------------|------------|------------|
| Component | Enter | Verify | Exit | | |
| System | | Enter | Verify- | Exit | |
| Integration | | | Enter | Verify | Exit |

This table shows an idealized case. Each test phase goes through three cycles, with the entry criteria for the next phase met after the first cycle for the previous phase. The first cycle of each phase identifies some number of “must-fix” issues, but not enough to violate the entry criteria for the next phase. A verification build, with fixes, is made, and the test suite is rerun. At this point, perhaps only one or two issues remain for that suite, so a final build, with fixes, is made, and the suite rerun a third time, confirming the fixes and meeting the exit criteria for that phase.

Theoretically, testing may require only one cycle, if all three test suites fail to identify any “must-fix” issues. (One proceeds directly from component to integration to system testing without a new build, since no issues exist and no new build is required.) In practice, this seldom occurs.

We generally had three phases of test that included:

- Unit/Component and String test: This responsibility lay mainly with the development organization.
- System Test which the Test Team was accountable for. This was a six-week period with three two-week test cycles each.
- Integration Test which the Test Team was also accountable for. This was a six-week period with four cycles within.
-

Acceptance Test

When we arrived, acceptance test occurred by selecting a test city and deploying the latest release of the product there. After some length of time, the product, with any corrections arising from the acceptance test phase, exits acceptance test. The installation then proceeds.

This process proved to be costly, in that the Test City would occasionally receive builds that would cause great customer dissatisfaction thus impacting revenue in these test markets. This process was never formalized either. Some amount of “actual use” is difficult to reproduce in a test setting, so ad hoc testing by end users covers areas of the product that the test team will not. However, it is important that acceptance testing also follow a standardized test suite, including the following elements:

- Testing of the new functions, using the release notes and the requirement documents.
- Testing of critical operations; i.e., those functions that, were they not to work properly, could result in significant financial loss.
- Testing of typical operations; i.e., those functions most-frequently performed that, were they not to work properly, could result in inconvenience and inefficiency.

- Testing of recurring errors that surface from time to time; i.e., instability, occasional freezes under specific conditions, etc.

In addition, testing according to a standardized test suite entails allocating sufficient resources and time to complete execution. Therefore, acceptance test schedules should not derive solely from external priorities and deadlines.

At the end of the acceptance test, a “closed loop” process occurs against the requirement database. In other words, those requirements and issue reports that the end users acknowledge as resolved by the latest release should be marked as “closed”.

Documentation

We developed and used the following five documents:

- **Maintenance Test Plan.** This overall plan addresses definitions, scope, risk management, failure mode effect analysis results, test configurations and environments, test execution process, test phases, entry and exit criteria for each phase, release management, issue tracking and analysis, status measurement, and test/development communication processes.
- **Component Test Suite.** This document will define a set of test cases to exercise individual functional components of the product.
- **Integration Test Suite.** This document will define a set of test cases to exercise the interfaces and connections between individual functional components of the product.
- **System Test Suite.** This document will define a set of test cases to exercise the overall behavior of the each separate component.
- **Version Status Report:** This document will allow one to view at a high level what was tested, what passed testing and what did not as well as the bug ids of any issues that have been opened against the system under test.

After the development of these documents, they undergo a peer review, including development and test staff. Then, the test team thoroughly tests the test suites themselves. The test team reviews each document during the specification and development phases for each release. Any enhancements or changes to the plan, processes or suites necessary to handle testing of the new release occur at that time.

Client Server GUI Testing

The Client server GUI Testing was one of the most successful projects in terms of early bug find/bug fixing. It was obvious from the outset that this project was one of the more organized in terms of design and clarity of specifications. This alone made the job of testing this application that much easier since it was clear to the test technicians what they were supposed to be testing and how. A prototype of the interface was also made available at an early stage so that the testers could start early with the automation in that the screen/window declarations and some of the primary functions could be created.

This application is created using Vantive, a third party supplier, but did necessitate many modifications to the Vantive code as we had many custom types of functionality required that Vantive simply did not offer. This application is running on Compaq workstations with Windows NT loaded and a minimum of application software and thus did not require extensive testing of different configurations. In past projects that Torsten Baumann has worked on he has found that testing for all possible configurations of workstations, was next to impossible since there was always someone that was running another application and thus system files may have been of different versions and the application would not work. This made our task that much easier as well, since all client desktops had the same file-set/image.

The server side resides on a Sun-UX box, which contains the Vantive application server, the Sybase database, Jaguar and Tibco.

The application under test was your basic Customer Service application with the regular features such as Member registration, Billing, Purchasing, Trial Offers, etc....

The Test Tool of choice for this application was QA Partner. The reason for this choice was quite simple actually. It was determined early in the project that this would be a GUI interface and a capture/playback tool would be used as it would be easier to train new testers with a small programming background in its use. Three of the Test Engineers that were hired on for this project also had experience using QA Partner in past projects and thus would not have to learn a new tool. Since Vantive is written with VBA and QA Partner can find most objects without much difficulty, we chose this approach. We generally would test the server by means of the client and QA Partner with all the piece parts running or stubs in place for those that were not yet ready for test.

Client Side Testing

There were been many discussion as to the approach we should be using. Should we do Client Side Testing or should we go right on the server and create tests there. In Keeping with the Black Box approach, the decision was made to go with Client Side Testing due to many different reasons some of which we will discuss below.

Traditionally one would think that creating tests from the client side will be easier simply for the reason that you may not need programmers as testers since the capture/playback did it all for you. This is not the reason we chose to go Client Side. We have found that what is generally known, as capture/playback in various test tools is not as simple as what some make it out to be. There is almost always a need for someone with some programming experience to make some modifications to actually make a useful test case. The main benefits we found were:

- The User Interface of the application will be tested at the same time.
- It was decided early in the project that the User interface will be frozen thus the test script maintenance would be kept to a minimum.
- This way we would be testing the application from the user perspective and results will be the same as those experienced by the users once the project went into production.
- We had the budget available for as many workstations as we needed to test that the system can handle certain loads. There was no concern that we would not have the hardware available to mimic the actual production environment.
- The middle-ware is exercised as well. Since Server Side Testing would not allow for the middle-ware to be tested under real-world conditions.

For this client server testing, testing scenarios followed a particular flow. The following incremental progression was used:

- Perform basic functional testing on a single client application so as to have those features needed for stress testing working effectively.
- Perform functional and concurrency testing on two remote client applications.
- Perform load testing of the server with a configuration large enough to stress the server (i.e. 100 users requirements)
- Perform extensive functional testing of all features.
- Perform regression testing.

Computer Telephony Interface Testing

The Configuration of the System under test is described as follows:

- CM server – NT server based
- CM client – NT workstation based
- CM IVR – UNIX based
- Servers application are C++
- Client application is Visual Basic

The CT System Operation is described as follows:

- IVR Member submits content via telephone
- IVR sends voice file and member information to CM server to access member profile
- CM server sends member profile and voice file information to CM-IVR
- CM IVR routes content to phone at CM client
- CM client displays member profile when call is answered
- CM client assigns status (approve/reject), call is disconnected, CM client is set to ready state
- CM server will also send status information to both Customer Service Application and the Home region IVR.

Manual Testing

Initial testing was done primarily in a manual fashion so as to gain familiarity with the application and to verify object content for those objects not recognized by the test tool. This involved using a utility (batch script calling C++ exe with DLLs), developed by the development team which was used to simulate incoming IVR telephone calls so that the Test Team could then simulate real-time calls into the CT system. This same DLL was used effectively to generate a background load on the application during Business Acceptance Test as well.

Manual Testing of this and all applications continues today, as one of our mandates is to ensure that the user will get the experience they crave when using the applications.

For this entire project (i.e. all subsystems) we have developed a test-case template that the engineers/testers will fill out in an easy to read sort of way and then at this point it was quite simple to have temporary workers and/or iMG employees execute the test-cases. The process was as follows:

- The testers (temporary worker, iMG employee) would be placed into teams of two (one playing IVR member and the other playing customer service representative).
- They would then be given a test-case, go back to a desk where a PC with the call center applications loaded, a telephone so they could dial the IVR as well as a second Lucent phone that was connected to the switch to route incoming calls from the IVR existed.
- The Tester would then begin doing each action item described in the test-case until the system under test would not function as described in the test case provided.
- At this point they would call a test technician who would validate the functionality, isolate the defect if existent and enter a bug report if required.
- The Testers would then go back and get another test case and so on.

See figure 2.0 for example of test case.

| Action ID | Subsystem | Step | Person initiating the step | Status | Bug ID |
|-----------|-----------|-----------------------------|----------------------------|--------|--------|
| 1 | IVR | Dial in | | | |
| | IVR | Listen to Welcome message | | | |
| | IVR | | | | |
| 2 | IVR | Skip the changes | | | |
| 3 | IVR | Press # at the login prompt | | | |
| 4 | IVR | Choose Guest access | | | |
| 5 | IVR | Choose Female | | | |

| | | | | | |
|----|-----|--|--|--|--|
| 6 | IVR | Choose Intimate | | | |
| 7 | IVR | now at main menu | | | |
| 8 | IVR | Choose access the info center | | | |
| 9 | IVR | Choose exit | | | |
| 9 | IVR | Choose Record/Listen to success stories | | | |
| 11 | IVR | Choose record testimonial | | | |
| 12 | IVR | Record the following voice message "Testimonial, Female Guest, Intimate, Reject" | | | |
| 13 | IVR | Choose accept valid testimonial. This posts the testimonial to the CM | | | |
| 14 | CM | CM monitors and rejects the testimonial | | | |

Figure 2.0

Automation development: Screen definitions

Those same utilities to simulate incoming calls that were discussed above were used to develop the automated test suite for this component. Some of the scenarios that we have lived through are:

- The Test Team was NOT part of the early design process. A problem encountered during this phase was that Visual Basic allows developers to not assign a handle to an object, this makes accessing objects for activities such as capturing text difficult if not impossible with automation tools. The lesson learnt by this is that a QA/QC or Test organization should be part of the design process, as retrofitting has impacted our schedules considerably due to this.
- If an object does not have a handle you may have to interact with it strictly by relative pixel location. Although this is not the optimal way to test it does work acceptably with objects such as buttons, check boxes, radio buttons, etc. Most tools will handle this acceptably well, if the GUI is sized so that the object is displayed.
- The CM client had several objects, buttons and text fields, which did not have handles; the buttons were interacted with by the above method. Text fields or boxes can not be; automation tools can not access text in an object they do not recognize. Unfortunately, one such text box had information related to the state of the GUI that was necessary to automate the testing. This required the developers to code additional utilities so that we can extract this information from other means besides the actual application.

Functional testing

- Because of the issues related to objects not having handles it was requested that the handles be added. Development felt that would negatively impact the schedule and possibly introduced more bugs. Their solution was to provide a window with the necessary status information when a command line parameter was set.
- The status window did provide the status information needed to automate the testing, but eventually we found it presented timing problems.
- Using utilities supplied by development we were able to simulate incoming content with profile information and telephone keypad key presses to test the CM client GUI functionality.
- The testing consisted of verifying the functionality of the various controls in the GUI and the proper update of the database.

Stress/Performance Testing

- Distributed processing – Host workstation connects to clients and sets up the client agent.
- A second workstation is set up to run a script that uses the DLL's to generate simulated calls. This script queues a specified number of calls per unit of time.
- When a call comes into a client the script running on the host communicates with the client to handle the call. The content was variously always approved, always rejected, or approved or rejected at the rate projected by marketing analysts.
- Initially, content was queued at a rate approaching development's projected maximum capacity. This led to problems, although the clients were handling content at an acceptable rate the servers quickly hung due to an inability to handle the load.
- Content queuing rates were lowered until the servers were able to handle the load. Then we slowly started increasing the rate until the problems started occurring again; this allowed development to find the problem.

Product Wide Integration Testing Using Multi-tools

Create a Test Software and Hardware Infrastructure That Allows Automated Testing of a Heterogeneous Enterprise Wide System

Tools Used:

- QA Partner for CT and CSA interfaces
- Perl, Tcl, Unix Shell Scripts for IVR Telephony interfaces
- TSFP (Test System Fake Pipe) (see figure 3.0)

Problems to overcome:

- The subsystems involved in the project run each on different platform using different operating systems. Each subsystem has a different kind of interface, based on a different set of paradigms, and a different type of test tools for test purposes.
- Testing must be non-intrusive; therefore the target applications are running in production mode (no debug and/or test configurations) as much as possible.
- The subsystems under test interact widely and continuously; therefore we must be able to test the system as a whole.

Solutions

- We must find a way to make the test drivers for each subsystem talk to each other outside the (sub) system(s) under test.
- The test feedback loop must be able to follow all the data transformations as the information flows across the system under test.

Implementation

- For each test thread, we establish a pool of data containing information about the state of each test driver involved as well as the associated test data. Each test driver is aware of the states of all the other test drivers involved, and is responsible to make correct test decisions based on this information.
- The integrated test applications use a state driven architecture:
 - The first test driver launches the tests and prepares the state information pool. It then drives its target subsystem through the programmed test flow, and posts the associated state transition info and test data to the state information pool. It pauses and waits for the next test driver to execute.
 - The second test driver was monitoring the state information pool. It senses the state transitions, drives its target subsystem through the programmed test flow, and posts the associated state transition info and test data to the state information pool. It then waits for the next test driver to execute.

- For the purposes of this presentation, we assume there are only two subsystems under test involved. The first test driver senses the state transition, wakes up and drives its subsystem into a “verification of the results” operation. It posts the info and exits.
- The second test driver senses the state transition, wakes up, and executes a similar operation for its target subsystem and exits.

Other Considerations:

- Each integrated test thread is independent. Any number of them, subject to resource availability, can execute simultaneously, enabling us to execute realistic test scenarios for the system under test.
- The implementation is very flexible. We use a .ini file structure with a general section, and a particular section for each subsystem (and its test driver) under test. Any type of test tool that can access text files over a TCP/IP network can be part of this test harness.
- The implementation is highly scalable. We can add any number of subsystems to our integrated test infrastructure by extending the state info pool with a corresponding section.

Test System Fake Pipe

The Test System Fake Pipe (TSFP) is used so that the telephony test driver and the client server test drivers can communicate with one another. Below you will find some sample code that locks, reads from and updates the TSFP as well as an example TSFP .ini file follows.

QA Partner sample code:

```
[-] type TsfpcMDData is record //read from *.ini
    [ ] string sGender
    [ ] string sProduct
    [ ] string sRegion
[+] type TsfpcMResults is record//write into *.ini
    [ ] string sAdSuccess
    [ ] string sGrSuccess
    [ ] string sIVRState
    [ ] string sCSAState
    [ ] string sCMState
    [ ] string sStartTime
    [ ] string sFinishTime
[+] // SetUpCMActions () comments
    [ ]
//*****
*
    [ ] //
    [ ] // SetUpCMActions ()
    [ ] //
    [ ] // Purpose:
    [ ] // Reads the fake pipe .ini, and creates action file in QAP host workstation
    [ ] //
    [ ] // Usage:
    [ ] // SetUpCMActions (sTSFPFile, sTestPath)
    [ ] //
    [ ] // Parameters:
    [ ] // sTSFPFile - lstring; contains name of .ini file to read for test parameters
    [ ] // sTestPath - string; path where the actions file is to be written
    [ ] //
    [ ]
//*****
*
```

```
[+] TsfpcMDData SetUpCMActions (string sUAN, string sTSFPPath)
    [ ] string sInFileName
    [ ] FILEINFO fFile
    [ ] list of FILEINFO lfInfo
    [ ] IFILE ifTSFPFile = IFile ()
    [ ] TsfpcMDData tParms
    [ ] string sPattern
    [ ]
    [ ] sPattern = "*" {sUAN} *.*"
    [ ] PRINT (sPattern)
    [-] // do
        [ ] lfInfo = SYS_GetDirContents (sTSFPPath)
        [ ] Print (sTSFPPath)
        [-] Print (lfInfo)
            [-] for each fFile in lfInfo
                [-] if ( !fFile.bIsDir )
                    [-] if ( MatchStr (sPattern, fFile.sName) )
                        [ ] ifTSFPFile.Open (sTSFPPath,fFile.sName)
                        [ ] // Get test parameters from the .ini file
                        [ ] tParms = GetTsfpcMDData (ifTSFPFile)
                        [ ] Print (tParms)
                        [ ] return tParms
                    [-] // except
                        [ ] LogError("File for UAN: {sUAN} not found")
        [+] return tParms
    [ ]
[-] // GetTsfpcMDData () comments
    [ ]
//*****
*
    [ ] //
    [ ] // GetTsfpcMDData ()
    [ ] //
    [ ] // Purpose:
    [ ] // Reads a fake pipe .ini, gets the CM parameters from that .ini file and returns that
    [ ] // information
    [ ] //
    [ ] // Usage:
    [ ] // GetTsfpcMDData (ifTSFPFile)
    [ ] //
    [ ] // Parameters:
    [ ] // ifTSFPFile - IFILE; name of .ini file to read for a test's parameters
    [ ] //
    [ ]
//*****
*
[-] TsfpcMDData GetTsfpcMDData (IFILE ifTSFPFile)
    [ ] //
    [ ] //Section Headers
    [ ] string sRegistration = "Registration"
    [ ] string sUpdateMemberInfo = "UpdateMemberInfo"
    [ ] string sAds = "Ads"
    [ ] string sGreetings = "Greetings"
    [ ] string sState = "State"
    [ ] string sData = "CMTime"
    [ ] //
```

```

[ ] //Values
[ ] string sGender = "Gender"
[ ] string sProduct = "Product"
[ ] string sRegion = "Region"
[ ] //Test
[ ] string sStartTime = "StartTime"
[ ] string sFinishTime = "FinishTime"
[ ] TsfpcMData tParms
[ ]
[ ] tParms.sGender = ifTSFPFile.GetValue(sData,sGender)
[ ] tParms.sProduct = ifTSFPFile.GetValue(sRegistration,sProduct)
[ ] tParms.sRegion = ifTSFPFile.GetValue(sRegistration,sRegion)
[ ] return tParms
[ ]
[-] // SetTsfpcMData () comments
[ ]
//*****
*

[ ] //
[ ] // SetTsfpcMData ()
[ ] //
[ ] // Purpose:
[ ] //  Reads a CM results file and then updates the .ini file
[ ] //
[ ] // Usage:
[ ] //  SetTsfpcMData (ifTSFPFile, sTestPath, sUAN)
[ ] //
[ ] // Parameters:
[ ] //  ifTSFPFile - IFILE; name of .ini file to update with a test's results
[ ] //  sTestPath - string; path where CM results files are located
[ ] //  sUAN - string; content ID for a test, used to generate CM results file name
[ ] //
[ ]
//*****
*
[-] boolean SetTsfpcMData (IFILE ifTSFPFile, TsfpcMResults tCMResults)
[ ] //
[ ] //Section Headers
[ ] string sAds = "Ads"
[ ] string sGreetings = "Greetings"
[ ] string sState = "State"
[ ] //
[ ] //Values
[ ] string sStartTime = "StartTime"
[ ] string sFinishTime = "FinishTime"
[ ] string sAdSuccess = "CMSuccess"
[ ] string sGrSuccess = "CMSuccess"
[ ] string sIVRState = "IVR_State"
[ ] string sCSAState = "CSA_State"
[ ] string sCMState = "CM_State"
[ ] //
[ ] //Test
[ ] string sSection
[ ] boolean bDone = False
[ ]
[-] if tCMResults.sAdSuccess != NULL

```

```

    [ ] ifTSFPFile.SetValue (sAds, sAdSuccess, tCMResults.sAdSuccess)
    [ ] ifTSFPFile.SetValue ("CMTimeAD", sStartTime, tCMResults.sStartTime)
    [ ] ifTSFPFile.SetValue ("CMTimeAD", sFinishTime, tCMResults.sFinishTime)
[-] if tCMResults.sGrSuccess != NULL
    [ ] ifTSFPFile.SetValue (sGreetings, sGrSuccess, tCMResults.sGrSuccess)
    [ ] ifTSFPFile.SetValue ("CMTimeGR", sStartTime, tCMResults.sStartTime)
    [ ] ifTSFPFile.SetValue ("CMTimeGR", sFinishTime, tCMResults.sFinishTime)
[ ] ifTSFPFile.SetValue (sState, sIVRState, tCMResults.sIVRState)
[ ] ifTSFPFile.SetValue (sState, sCSAState, tCMResults.sCSAState)
[ ] ifTSFPFile.SetValue (sState, sCMState, tCMResults.sCMState)
[ ]
[ ]
[-] while ( !bDone )
    [-] if ( ifTSFPFile.SetLock ("CM") )
        [ ] ifTSFPFile.Close ()
        [ ] ifTSFPFile.iStatus = DONE
        [ ] ifTSFPFile.ClearLock ("CM")
        [ ] bDone = TRUE
        [ ] print ("result = {tCMResults.sStartTime} - (clear lock)")
    [+] else
        [ ] sleep (2)

[ ]
[ ] return bDone
[+] boolean GotResults (string sTestPath, string sFileName)
    [ ] boolean bGotResultsFile = FALSE
    [ ]
    [+] if ( hHost -> SYS_FileExists ("{sTestPath}\{sFileName}.out") )
        [ ] bGotResultsFile = TRUE
    [ ] return bGotResultsFile
[ ]

```

Perl Sample Code:

A wrapper module exists that we call StateDriver written in Perl which is used for fakepipe interaction. It's usage by a script as follows:

```

# Tells script to use statedriver module
use StateDriver;

# loads the current tsfp_filename into an array
$tsfpFiles[$index] = $tsfp_filename ;

# Sets the TSFP filename in the statedriver module for all subsequent calls
&StateDriver::SetTSFPFileName(@tsfpFiles) ;

# Asks statedrive for UAN field under header Data (index is the index in the array and 60 is the # timeout)
my ($uan)= &StateDriver::GetKey("Data", "UAN", $index, 60) ;

# Tells Statedriver to change State information to IVR_State=GO, CSA_State=WAIT and #
CM_State=FINISHED
&StateDriver::SetState("GO", "WAIT", "FINISHED", $index, 60);

# Tells Statedriver to make PreTestCSABalance field to 50 under header Data
&StateDriver::UpdateKey("Data", "PreTestCSABalance", "50", $index, 60);

```

These calls make TSFP interaction easier and allows code to be cleaner as TSFP calls can become lengthy and make code look messy. Two of the module procedures follow.

2 procedures from StateDriver.pm

```
sub StateDriver::GetKey {

    my($section) = shift(@_);
    my($keyValue) = shift(@_);
    my($ini_selection) = shift(@_);
    my($time_out) = shift(@_);

    my($count) = 0;
    my($tsfp_file) = $stfTSFP_File[$ini_selection];

    my($tsfp_fp, $tsfp_head);

    until(($tsfp_fp = &tsfp::tsfp_get_lock($tsfp_file)) ||
        ($count++ ge $time_out)) {
        print " Waiting On Fakepipe .....","\n" ;
    } ;

    $tsfp_head = &tsfp::tsfp_load($tsfp_fp) or ( (&tsfp::tsfp_clear_lock($tsfp_head, $tsfp_file, $tsfp_fp))
    &&
        ( return ) );
    my($valueOnTsfp) = &tsfp::tsfp_retrieve_key($tsfp_head, $section, $keyValue);
    &tsfp::tsfp_write($tsfp_head, $tsfp_fp);
    &tsfp::tsfp_clear_lock($tsfp_head, $tsfp_file, $tsfp_fp);
    return $valueOnTsfp;
}
```

```
sub StateDriver::UpdateKey {

    my($section) = shift(@_);
    my($keyValue) = shift(@_);
    my($sttResult) = shift(@_);
    my($ini_selection) = shift(@_);
    my($time_out) = shift(@_);

    my($count) = 0;
    my($tsfp_file) = $stfTSFP_File[$ini_selection];

    my($tsfp_fp, $tsfp_head);

    until(($tsfp_fp = &tsfp::tsfp_get_lock($tsfp_file)) ||
        ($count++ ge $time_out)) {
        print " Waiting On Fakepipe .....","\n" ;
    } ;

    $tsfp_head = &tsfp::tsfp_load($tsfp_fp) or ( (&tsfp::tsfp_clear_lock($tsfp_head, $tsfp_file, $tsfp_fp))
    &&
        ( return ) );
    print STDERR "$section\t$keyValue\t$sttResult", "\n" ;
    &tsfp::tsfp_update_key($tsfp_head, $section, $keyValue, $sttResult);
    &tsfp::tsfp_write($tsfp_head, $tsfp_fp);
}
```

```
&tsfp::tsfp_clear_lock($tsfp_head, $tsfp_file, $tsfp_fp) ;  
1 ;  
}
```

Figure 3.0: TSFP.ini template

```
[Registration]  
RegistrationMethod=  
Product=  
Region=  
VoiceApproval=  
ServiceAgreement=  
UANGeneration=  
VoicePrintWaitTimeout=  
CMVoiceApprovalSuccess=  
IVRSuccess=  
CSASuccess=  
CSAStartTime=  
IVRStartTime=  
CMStartTime=  
CSAEndTime=  
IVREndTime=  
CMEndTime=  
CMWarn=  
CSAWarn=  
IVRWarn=  
IVRError=  
CSAError=  
CMErrror=
```

```
[UpdateMemberInfo]  
Method=  
ChangeUAN=  
ChangePPC=  
ChangeZIP=  
ChangeUANPPC=  
NewUAN=  
NewPPC=  
NewZip=  
IVRSuccess=  
CSASuccess=
```

```
[Content]  
Segment=  
ExpectedResult=
```


Timeout=
CMSuccess=
CSASuccess=
IVRSuccess=
CSAStartTime=
IVRStartTime=
CMStartTime=
CMEndTime=
CSAEndTime=
IVREndTime=

[State]
IVR_State=
CSA_State=
CM_State=

[Data]
Gender=
PostalZip=
BirthDate=
MemberUAN=
UAN=
PPC=
PreTestIVRBalance=
PreTestCSABalance=

Management Considerations

In addition to the technical challenges facing the IMG Test Team, a number of complex management considerations arose. Many of these arose in three main areas: logistics, staffing the team, and tracking bugs and test cases. The following paragraphs discuss some of these challenges and how we managed them.

Logistics

The IMG facilities are located in two areas of the Greater Toronto Metropolitan Area. The headquarters are on King Street West in downtown Toronto, just blocks from the financial district. The call center, though, is located in Etobicoke near the Lester Pearson Airport. These two locations are about fifteen kilometers apart, which is usually fifteen minutes' drive during non-peak traffic hours. Because of the hardware requirements of the testing (see below), some testing took place in each location. This led to a distributed testing approach.

Having two test teams separated by only fifteen kilometers may seem trivial. It would appear less challenging than some test efforts one of the authors, Rex Black, has managed, which spanned the Pacific Ocean and involved participants in Taiwan, Japan, Northern California, Southern California, and Utah. However, in this case, the expected results of a test in one location depended heavily on the testing and status in the other. For example, if an IVR server went down on King Street, this could affect what the testers were seeing at Etobicoke. Likewise, CSA server crashes could delay and even lose data sent to the IVR server via the Send/Listen mechanisms provided by TIBCO.

The information transfer challenges created by this tight coupling of components were made even more difficult by the fact that we performed both manual and automated testing. The manual testing involved large teams who had to be able to start and stop within moments. The automated tests, likewise, needed

precision control based on an accurate picture of the current status of all the hardware, software, and network components. Single points of failure and latency made the status even less clear-cut. The complex hardware, software, and network test environment simulated an even more complex hardware, software, and network field environment. We had a pair of IVR servers acting as deployed systems with two more IVR servers driving them with automated tests downtown, and a handful of various servers and telephony components, along with almost 100 client systems, at the call center. The IVR servers ran Unix, as did some of the telephony and database components. However, other servers and the client desktop systems ran Windows. Custom, customized, and COTS software ran throughout the environment. Finally, the network itself was a custom amalgamation of Ethernet, ISDN, and WAN elements. At least we only had a couple IVR servers located in the same town as the call center. The deployed system would have dozens of IVR servers, located singly and in pairs throughout North America, Australia, and, ultimately, Europe.

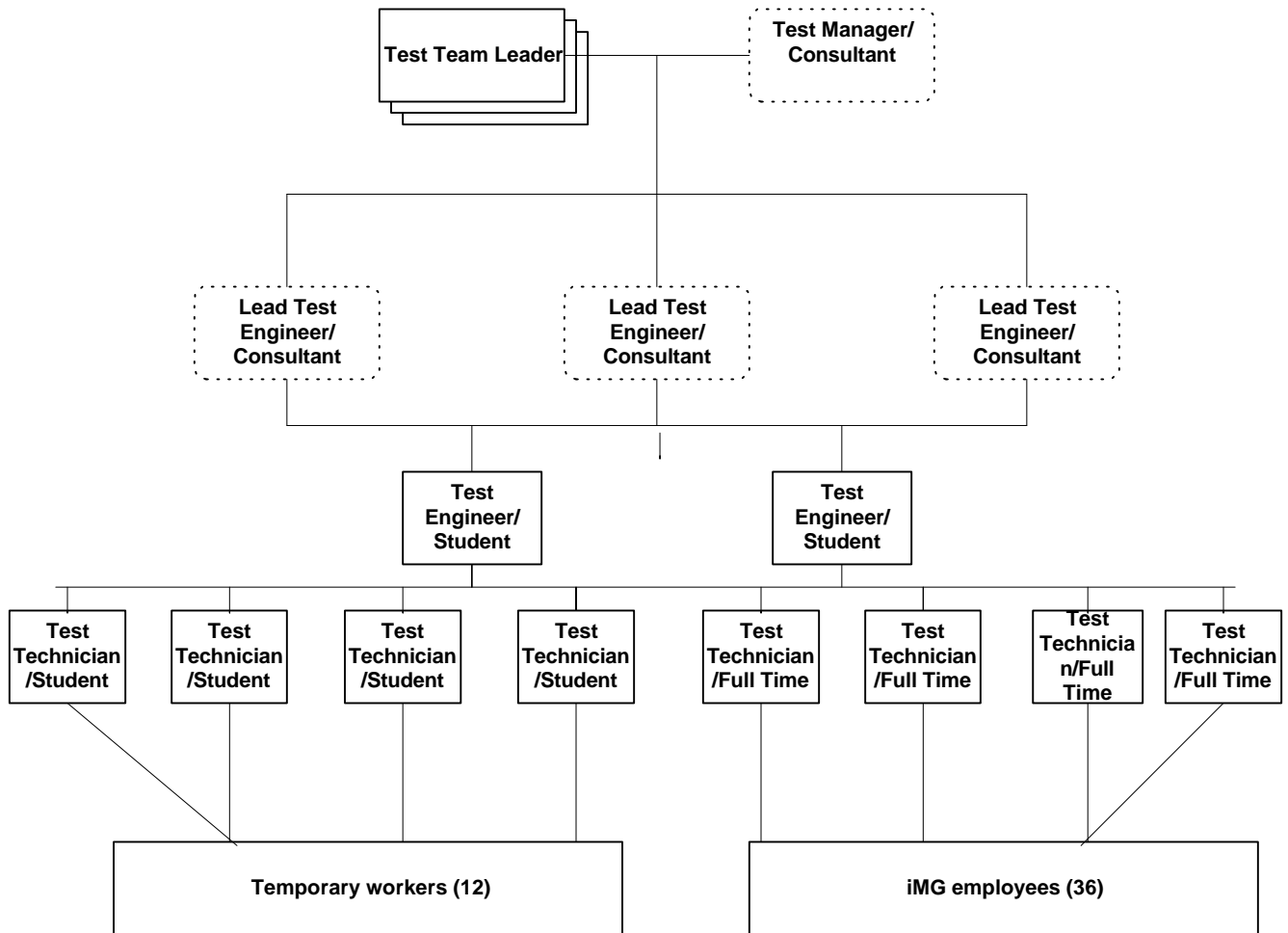
Information transfer challenges were dealt with three ways. First, the test team managers and the engineers spent a fair amount of time at each site, regardless of their direct responsibilities. This led to rapid resolution of communication breakdowns and sympathy for each other's problems. Second, cell phones were issued to key players, individual contributors and managers, to make sure that escalation of problems and communication hurdles could happen instantly. Finally, shared bug and test tracking tools allowed each team to see the concrete results of the other's testing, even between detailed test status review meetings. The network complexity issues were dealt with by careful monitoring of status and instant escalation to outside system administration support when problems arose.

Test Team Size and Composition

The team was composed of a mixture of IMG full-time employees, IMG part-time employees, and RBCS consultants. About fifty people participated in the test effort at one point, though the core team was comprised of about fifteen people. The RBCS consultants between them have over 50 years of test experience, while the IMG employees tended to be new to software development projects or testing specifically. Therefore, the consultants served as both implementers and transferors of knowledge. The exception to the IMG test team experience levels was the permanent IMG test manager, Torsten Baumann, who has a degree, hands-on management experience, and software testing experience. He brought his familiarity with test automation and relational databases to bear in a technical capacity, too, designing and implementing the Customer Service Application and Customer Data Repository tests, both manual and automated. Rex Black, the RBCS consultant test manager, got the process going initially, developing a budget, schedule, and plan for System and Integration Test. Once in Integration Test, he served primarily to assist Torsten in his efforts.

In addition to the test management professionals, a total of five test engineers worked on the project. One, Barton Layne, worked specifically on designing and implementing IVR testing. Another, Gordon Page, worked primarily on Content Management testing, especially automation of these tests and "marrying" that automation—as described in this paper—to the automation on the IVR side. Yet another test engineer, Serban Teodorescu, worked on the integration testing, addressing not only the intercommunicating pieces but also the testing of the "glue"—the Publish/Subscribe mechanism—that allowed the components to talk. All three of these engineers are RBCS consultants. Two more test engineers—at the time students at the University of Toronto—worked creating test tools such as the telephony load and traffic generator, CallSim, and the CallSim Test System Fake Pipe library.

Supporting both automated and manual testing were eight test technicians, a mixture of students and full-time professionals. At the peak of the manual testing, about a dozen temporary workers were brought in from a local staffing agency to run tests over a sixteen-hour-per-day period. Also, about two or three dozen IMG employees augmented the testing. Some of the Customer Service Representatives performed their jobs but with simulated IVR users who were actually testers. Other IMG employees simulated IVR use to generate load and to surface interface usability problems.



Such a test team may strike some as eclectic, but the approach is an unfortunately underutilized but excellent way to bootstrap a test organization into existence. Getting an independent test team off the ground is tough, and around 25 percent of such groups are disestablished within two years of their creation. It's important to build a solid foundation for future growth and to produce visible results immediately. By using seasoned professional test consultants, the test team was able to avoid many dangerous pitfalls, perform valuable testing right away, and design a solid test system architecture for the team going forward. Once the initial hurdles were passed, the consultants phased out, leaving behind the experienced test manager and a team of less experienced but now battle-proven testers who had now seen the job done properly and had inherited a professional-quality test system.

We did suffer from some drawbacks related to the non-test staff participation, namely trouble with writing decent bug reports. The test engineers and technicians used a review process to improve bug reports, but they had a common set of expectations about what constituted a "good" bug report, including detailed steps to reproduce and proper isolation. Amateur testers had no idea how to communicate problems effectively to development. In cases where the test team supervised and reviewed the amateurs' test results, considerable time was spent getting the reports in a usable format. In cases where the test team did not screen bug reports, the reports were generally useless to development.

Bug and Test Tracking

An important challenge facing any test organization is the orderly tracking and management of the tests performed, the bugs found, and the risks to quality addressed and abated by these results. Various commercial-off-the-shelf-software tools exist to perform these roles, but, since none were perfect fits for the complex, heterogeneous environment discussed in this paper, IMG decided to license a customizable solution from RBCS. The three components of this solution provide an integrated, quantitative test management system for bugs, tests, and risks to quality. As part of the test effort, the test team adapted these tools to handle these essential tasks.

The foundation of any well-engineered test system is a risk-based approach to test development. Which problems, if they occur, will seriously harm the users' experiences of quality, and which other bugs constitute mere nuisances? A quantitative approach to answer this question was provided by the Failure Mode and Effect Analysis technique.¹ The toolkit extended this technique by measuring test case coverage against the risks to quality identified through FMEA, and, by applying numerical ratings to coverage, giving management a yardstick for benchmarking quality testing.

As tests are developed, one must have a standard approach to documenting them. Once execution starts, one must track the results and status of each test case, and be able to report summary information on all the test suites across each test effort. A three-tiered method, based on Excel worksheets, provided information at the test step, test case, and test suite level. The lowest-level documentation provided the exact steps required for the test technicians to set up, run, and tear down each test case. At the higher levels, this information was summarized for precise management control and presentation. To measure quality, test cases were assigned pass, block, or fail status initially, with blocked tests eventually being run, passed or failed, and failed tests, when the underlying failure was addressed, becoming closed. Other information, such as the tested hardware, software, and network configuration variables, was also captured.

The final, but in many ways most important, component was the bug tracking database. This was a distributed system, with a graphical, VBA-driven, Access-based front-end available on each development team participant's desktop, and the data tables stored in a universally accessible area of the network. File read and write abilities were controlled based on the permissions of the table and the Access workgroup file, and on the user groups: developer, manager, tester, and so forth. The database was state-driven, supporting review, rejected open, assigned, test, closed, deferred, and reopened states, with appropriate ownership assigned in each state. The database also supported the generation of various reports, project metrics, and charts such as the bug summary and the opened/closed graph.

Conclusion

In this paper, we have introduced the reader to various approaches that Interactive Media Group find valuable to successfully test one of the most complex test projects the authors have worked on. From simply testing a Call Center application, to testing an integrated telephony application, to testing a CT application, to testing for data quality and reporting, to testing all these components together, we hope that our experiences are as valuable to the reader as it was to the authors.

From Unit Component Test right through to Customer Acceptance Test, the most valuable lesson learned by the authors is that inter-team communications be that by way of an issue tracking log, e-mails or telephone, is invaluable. Also, we hope that the learnings and explanations provided with respect to the Test System Fake Pipe are found as useful as we have found them to be. This is a simple approach to a very complex test project that can be used anywhere and with any tools.

¹ For more information on applying this technique, see *Failure Mode and Effect Analysis*, by D. H. Statamis, or, for a software-specific discussion, see *Managing the Testing Process*, by Rex Black, one of the authors of this paper.

Recommended Readings

Rex Black: *Managing the Testing Process* (1999), Microsoft Press, Seattle, WA.

Boris Beizer: *Software System Testing and Quality Assurance* (1996), International Thomson Computer Press, Boston, MA.

Kelly C. Bourne: *Testing Client/Server Systems* (1997), McGraw-Hill, New York, NY.

Roger S. Pressman: *Software Engineering A Practitioner's Approach*, McGraw-Hill, New York, NY.

Biographies

Torsten Baumann

Torsten Baumann has spent five years in the Software Testing and Quality Assurance field. He is currently the Test Team Leader at Interactive Media Group, a leading international telephony applications company, based in Toronto, Canada. Prior to this he worked as a Software Test Analyst and Team Leader of Testing/Applications Development with Speedware corporation in Montreal, Canada. His work has taken him to the U.S, Germany, Switzerland and Canada. He is currently working on a book to be titled, *Manual Testing or Test Automation: What is the correct approach?*

At Speedware corporation, Mr. Baumann was the primary responsible for the testing of an Object Class Library, a 4GL Web Development tool, as well as several Case Tools for Visual Basic and 4GL. By use of Test automation he created several test suites to cover the above-mentioned products. He recently has been working at iMG in managing and automating the testing process of a complex, multi-faceted IVR Telephony/Client Server project.

Mr. Baumann attended Concordia University's Bachelor of Commerce Program, as well as graduated from John Abbot College's Programmer/Analyst program. He is currently pursuing certification with the Quality Assurance Institute as well as his Masters of Business Administration.

Rex Black:

Rex Black has spent sixteen years in the computer industry, with twelve years in testing and quality assurance. He is the President and Principal Consultant of Rex Black Consulting Services, Inc., an international software and hardware testing and quality assurance consultancy. His clients include Dell, SunSoft, Hitachi, Motorola, Tatung, NetPliance, General Electric, Pacific Bell, IMG, Renaissance Worldwide, DataRace, Omegabyte, Strategic Forecasting, and Clarion. His work with these clients has taken him to Taiwan, Hong Kong, Japan, the U.K., Canada, Germany, Holland, France, Spain, Switzerland, and Italy, along with locations throughout the United States. He authored *Managing the Testing Process*, published by Microsoft Press in its Best Practices series.

Before becoming a consultant in 1994, Mr. Black worked as Quality Assurance Manager at Locus Computing and IQ Software, doing operating system testing for IBM and database and data warehouse query and analysis tool testing. He also spent three years as a Project Manager at XXCAL-now owned by National Technical Services-a computer testing lab. Prior to that, he worked as a programmer and system administrator.

Mr. Black holds a B.Sc. in Computer Science and Engineering from UCLA. He belongs to the Association for Computer Machinery and the American Society for Quality.

Serban Teodorescu:

Serban Teodorescu holds a Masters Degree in Electrical Engineering from the Polytechnic Institute in Bucharest, Romania from 1989 as well as a Computer Science Certificate from Herzing Institute in Montreal.

Before entering the software industry he worked in nuclear power plants automation in Romania (1989-1990) as well as Locomotive drives automation in Holland(Netherlands) (1990-1993). Here he decided to enter the computer industry as a Windows System Administrator in the Netherlands from 1991

to 1993. Mr. Teodorescu then consulted as a Database developer from 1993 to 1996 after which he became a Programming instructor in Montreal. It was at this point that he pursued his career as a Software Test Analyst with Speedware Corporation in Montreal. He is currently working as a consultant with Interactive Media Group with the Test Team.

Gordon Page:

Gordon Page has worked in the software industry for 21 years as a developer and QA/test engineer. For the past 6 years he has worked in Quality Assurance and Test for IQ Software, Intersolv PVCS, MedicaLogic, BMC Software, and as a QA consultant specializing in GUI test automation.

The authors of this paper can be reached in any of the following ways:

Torsten Baumann

Phone: +1 (905) 816 0074

Fax: +1 (416) 640 1905

E-mail: TBaumann1@compuserve.com
Torsten_baumann@interactivemedia.com

Mail: Torsten Baumann
905 King Street West Suite 500
Toronto, Ontario MGK 3G9

Rex Black:

Phone: +1 (210) 696-6835

Fax: +1 (210) 696-8788

E-mail: Rex_Black@RexBlackConsulting.com

Mail: Rex Black
Rex Black Consulting Services
7310 Beartrap Lane
San Antonio, TX 78249

Serban Teodorescu:

Phone: +1 (416) 2636300/3403

+1 (416) 677-1351

Fax: +1 (416) 263-6308

E-mail: steo@netcom.ca

Mail: Serban Teodorescu
905 King Street West Suite 500
Toronto, Ontario MGK 3G9

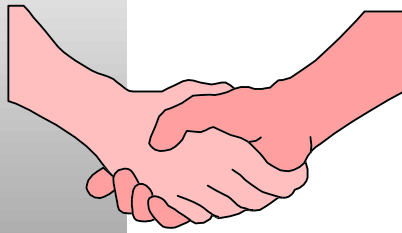
Gordon Page:

Phone: +1 (512) 656 8069 cell

E-mail: gordonpage@earthlink.net

Mail: Gordon Page
10610 Morado circle #1301
Austin, TX 78759

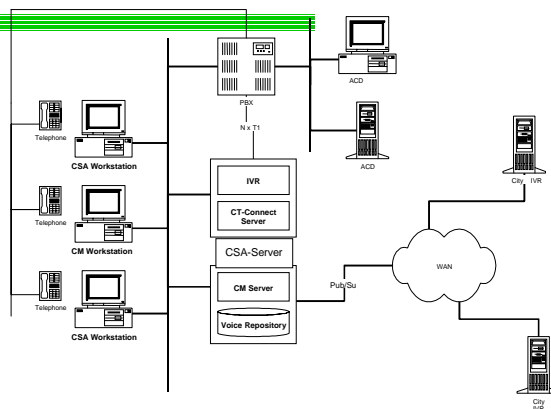
Integrated Testing of Telephony and Call Center applications



- **Torsten Baumann**
- **Rex Black**
- **Serban Teodorescu**
- **Gordon Page**

1

SUT Architecture



2

Discussion Topics

- **Client Server GUI (CSA)**
- **CTI Client Server (CM)**
- **IVR applications using multi-tools**
- **Product Wide Integration**
- **Management consideration**

3

Product Wide Integration Testing Using Multi-tools

- Create a Test Software and Hardware Infrastructure That Allows Automated Testing of a Heterogeneous Enterprise Wide System

4

Problems to overcome:

- The subsystems involved in the project run each on different platform using different operating systems. Each subsystem has a different kind of interface based on a different set of paradigms, and different type of test tools for test purposes.
- Testing must be non-intrusive, therefore the target applications are running in production mode (no debug and/or test configurations) as much as possible.
- The subsystems under test interact widely and continuously; therefore we must be able to test the system as a whole.

5

Solutions

- We must find a way to make the test drivers for each subsystem talk to each other outside the (sub)system(s) under test.
- The test feedback loop must be able to follow all the data transformations as the information flows across the system under test.

6

Implementation I

- **For each test thread, we establish a pool of data containing information about the state of each test driver involved as well as the associated test data. Each test driver is aware of the states of all the other test drivers involved, and is responsible to make correct test decisions based on this information.**
- **The integrated test applications use a state driven architecture:**
 - The first test driver launches the tests and prepares the state information pool. It then drives its target subsystem through the programmed test flow, and posts the associated state transition info and test data to the state information pool. It pauses and waits for the next test driver to execute.

7

Implementation II

- **The second test driver was monitoring the state information pool. It senses the state transitions, drives its target subsystem through the programmed test flow, and posts the associated associated state transition info and test data to the state information pool. It then waits for the next test driver to execute.**
- **For the purposes of this presentation, we assume there are only three subsystems under test involved. The first test driver senses the state transition, wakes up and drives its subsystem into a “verification of the results” operation. It posts the info and exits.**
- **The second test driver senses the state transition, wakes up, and executes a similar operation for its target subsystem and exits.**

8

Other Considerations

- Each integrated test thread is independent. Any number of them, subject to resource availability, can execute simultaneously, enabling us to execute realistic test scenarios for the system under test.
- The implementation is very flexible. We use an .ini file structure with a general section, and a particular section for each subsystem (and its test driver) under test. Any type of test tool that can access text files over a TCP/IP network can be part of this test harness.
- The implementation is highly scalable. We can add any number of subsystems to our integrated test infrastructure by extending the state info pool with a corresponding section.

9

Test System Fake Pipe (TSFP)

```
[Data]
Gender=
PostalZip=
BirthDate=
MemberUAN=
UAN=
PPC=
PreTestIVRBal=
PreTestCSABal=
PostTestIVRBal=
PostTestCSABal=
CCType=
CCNum=
CCExpMonth=
CCExpYear=
```

10

Test System Fake Pipe

```

[Registration]
RegistrationMeth=
Product=
Region=
UANGeneration=
IVRSuccess=
CSASuccess=
CSAStartTime=
IVRStartTime=
CMStartTime=
CSAEndTime=
IVREndTime=
CMEndTime=
CMWarn=
CSAWarn=
IVRWarn=
IVRError=
CSAError=
CMError=
    
```

11

Post Test Run Results File

| [] Fake Pipe | TestCase | Status | Error | Warning |
|-------------------------------|-----------------|--------|-------|------------------|
| [] 4900006675_11313_TSFP.ini | Registration | PASS | (CSA) | NA |
| [] | | PASS | (IVR) | NA |
| [] | Login | PASS | (IVR) | No Error |
| [] | UpdateMember | FAIL | (CSA) | CSA Error |
| [] | | FAIL | (IVR) | IVR Error |
| [] | InitialPurchase | PASS | (IVR) | Manual Authorize |

12

Logistics

- **Distributed test teams**
 - **Two locations about 15 kilometers apart**
 - Downtown Toronto
 - Airport area
- **Manual and automated testing approaches**
- **Complex test environment simulating an even more complex field environment**
 - **Hardware (dozens of servers, about 100 clients)**
 - **Software**
 - Unix (2 variants), Windows, call center, and telephony
 - Custom, COTS, and customized COTS
 - **Network (100BT E'net, PRI IDSN, WAN, PSTN)**

13

Test Team Size and Composition

- **IMG employees and RBCS consultants**
- **Two Test Managers**
 - One consultant
 - One employee
- **Five Test Engineers**
 - Three contractors for IVR, CM, and Integration
 - Two employees for IVR and Integration
- **Eight Test Technicians**
 - All contract IMG employees
 - About two each for IVR, CM, CSA, and Integration
- **About 12 temps for manual testing**
- **About 30 CSRs, other IMG employees ran manual tests**

14

Bug and Test Tracking

- **Integrated test management system from RBCS**
- **Quantitative management of risks to quality through FMEA failure and test coverage analysis**
- **Test tracking**
 - Customized Excel spreadsheets
 - Automatic test case and suite summarization
 - Test case details included in subordinate worksheets
- **Bug tracking**
 - Customized GUI, VBA-driven, Access-based database
 - Distributed to each tester's desktop w/ shared tables
 - State-based, with ownership for management to closure

15



How to Implement an Effective Requirements Management Process A Real Life Framework for Developing Your Own Approach

Dave Locke, Rational Software

A Rational Software white paper

Table of Contents

| | |
|---|-----------|
| Software and System Development in the Age Process | 1 |
| Why Manage Requirements?..... | 1 |
| What is a Requirement?..... | 2 |
| What is Requirements Management?..... | 2 |
| The Problems of Requirements Management | 2 |
| Requirements Management Skills | 3 |
| Key Skill 1: Problem Analysis | 3 |
| Key Skill 2: Understanding Stakeholder Needs | 4 |
| Key Skill 3: Defining the System | 4 |
| Key Skill 4: Managing the Scope of the Project..... | 5 |
| Key Skill 5: Refining the System Definition | 5 |
| Key Skill 6: Managing Changing Requirements..... | 5 |
| Important Requirements Concepts..... | 6 |
| Requirement Type | 6 |
| Cross-Functional Teams | 7 |
| Traceability | 7 |
| Multi-Dimensional Attributes | 7 |
| Change History | 9 |
| Putting Requirements Management to Work..... | 9 |
| Requirements Management Workflows | 10 |
| Workflow: Problem Analysis..... | 10 |
| The Activities in Problem Analysis | 11 |
| Workflow: Understanding Stakeholder Needs | 12 |
| Activities in Understanding Stakeholders Needs | 13 |
| Workflow: Defining the System..... | 14 |
| Activities in Defining the System..... | 14 |
| Workflow: Managing Scope | 15 |
| Activities in Managing Scope | 15 |
| Workflow: Refining the System..... | 16 |
| Activities in Refining the System..... | 17 |
| Workflow: Managing Requirement Change | 18 |
| Activities in Managing Changing Requirements..... | 19 |
| Summary | 19 |

Software and System Development in the Age Process

For most software and system* development teams, the 1990s have been process-intensive when compared to the more freewheeling days of the past. Standards for measuring and certifying effective software development process have been introduced and popularized. Many books and articles on software development process and related material on business process modeling and re-engineering have been published. Increasing numbers of software tools have helped define and apply effective software development process. The global economy's dependence on software accelerated in the decade, enabling development processes and improving system quality.

So how do we explain the high incidence of the software project failure today? Why are many, if not most, software projects still plagued by delays, budget overruns, and quality problems? How can we improve the quality of the systems we build as our businesses, national economies, and daily activities become increasingly dependent on them?

The answers, as always, lie in the people, tools, and processes applied to our profession. Requirements management is often proposed as a solution to the ongoing problems of software development, yet relatively little attention has been focused on improving the practice of this discipline.

This paper presents the elements of an effective requirements management process and highlights some of the obstacles to its successful implementation.

Why Manage Requirements?

Simply put, system development teams who manage requirements do so because they want their projects to succeed. Meeting their project's requirements defines success. Failing to manage requirements decreases the probability of meeting these objectives.

Recent evidence is supportive:

- The Standish Group's CHAOS Reports from 1994 and 1997 established that the most significant contributors to project failure relate to requirements.¹
- In December 1997, Computer Industry Daily reported on a Sequent Computer Systems, Inc. study of 500 IT managers in the U.S. and U.K. that found 76 percent of the respondents had experienced complete project failure during their careers. The most frequently named cause of project failure was "changing user requirements."²

Avoiding failure should be sufficient motivation to manage requirements. Increasing the probability of a successful project and other benefits of managing requirements may be equally motivational. The Standish Group's CHAOS report further established that managing requirements well was the factor most related to successful projects.

*Requirements management applies equally to software-only projects and to projects in which software is only a part of the end result or not included at all. For convenience, the paper will hereafter use the term "system" to mean any or all of these things. However, it is the abstract nature of software development, alone or in combination with hardware that complicates requirements management, and is therefore the primary focus of the paper.

1 CHAOS, The Standish Group International, Inc., Dennis, MA, 1994,1997

2 Computer Industry Daily, December 12, 1997

What is a Requirement?

The first step towards understanding requirements management is to agree on a common vocabulary. Rational defines a requirement as “a condition or capability to which the system [being built] must conform.” The Institute of Electronics and Electrical Engineers uses a similar definition.

Well-known requirements engineering authors Merlin Dorfman and Richard H. Thayer offer a compatible and more refined definition that is specific – but not necessarily limited – to software:

“A software requirement can be defined as:

- A software capability needed by the user to solve a problem or achieve an objective. A software capability that must be met or possessed by a system or system component to satisfy a contract, specification, standard, or other formally imposed documentation.”³

What is Requirements Management?

Since requirements are things to which the system being built must conform, and conformance to some set of requirements defines the success or failure of projects, it makes sense to find out what the requirements are, write them down, organize them, and track them in the event they change.

Stated another way, Requirements Management is:

- a systematic approach to eliciting, organizing, and documenting the requirements of the system, and
- a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system.

This definition is similar to Dorfman and Thayer’s and the IEEE’s “Software Requirements Engineering.” Requirements Engineering includes elicitation*, analysis, specification, verification, and management⁴. All of these activities are incorporated in the definition of requirements management presented here and taught by Rational Software. The difference lies mainly in the choice of the word “management” rather than “engineering.” Management is a more appropriate description of all the activities involved, and it accurately emphasizes the importance of tracking changes to maintain agreements between stakeholders and the project team.

The Problems of Requirements Management

So what might be difficult about a process intended to ensure that a systems conforms to the expectations set for it? When put into practice on real projects, difficulties come to light. Figure 1 displays the results of a 1996 survey of developers, managers, and quality assurance personnel. It shows the percentage of respondents who experienced the most frequently mentioned requirements-related problems.

³ Dorfman, M. and R. Thayer, Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1997 pp.79

*For those unfamiliar with the term “elicitation,” it is defined as the set of activities that teams employ to elicit or discover stakeholder requirements.

⁴ Dorfman, M. and R. Thayer, Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1997 pp.79

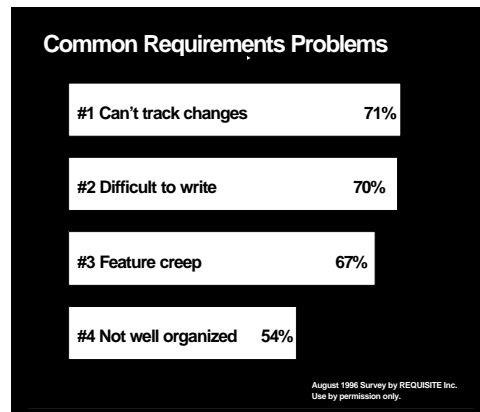


Figure 1: Common Requirement Problems

A more comprehensive list of problems includes:

- Requirements are not always obvious and have many sources.
- Requirements are not always easy to express clearly in words.
- There are many different types of requirements at different levels of detail.
- The number of requirements can become unmanageable if not controlled.
- Requirements are related to one another and to other deliverables of the process in a variety of ways.
- Requirements have unique properties or property values. For example, they are neither equally important nor equally easy to meet.
- There are many interested and responsible parties, which means requirements need to be managed by cross-functional groups of people.
- Requirements change.
- Requirements can be time-sensitive.

When these problems are combined with inadequate requirements management and process skills, and the lack of easy-to-use tools, many teams despair of ever managing requirements well.

Requirements Management Skills

To resolve the problems mentioned above, Rational encourages the development of *key skills*. These skills are presented below in what appears to be sequential order, but in an effective requirements management process they are applied continuously in varied order. Here they are presented in the sequence one would likely apply to the first iteration of a new project.

Key Skill 1: Problem Analysis

Problem analysis is conducted to understand business problems, target initial stakeholder needs, and propose high-level solutions. These acts of reasoning and analysis find “the problem behind the problem.”

During problem analysis, agreement is gained on a statement of the real problems and the stakeholders are identified. Initial solution boundaries and constraints are defined from both technical and business perspectives. If appropriate, the business case for the project analyzes return on investment that is expected from the system.

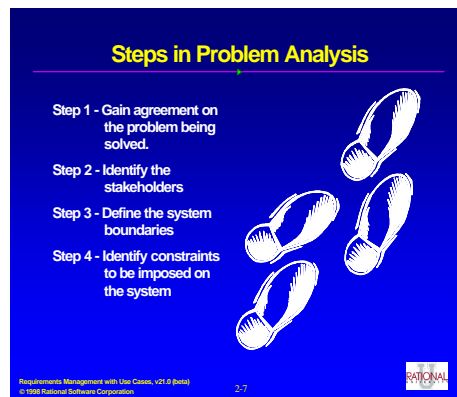


Figure 2: Steps in Problem Analysis

Key Skill 2: Understanding Stakeholder Needs

Requirements have many sources. They may come from anyone with an interest in the outcome of the project. Customers, partners, end users, and domain experts are some sources of requirements. Management, project team members, business policies, and regulatory agencies can be others.

It is important to know how to determine who the sources should be, how to get access to those sources, and how to elicit information from them. The individuals who serve as primary sources for this information are referred to as “stakeholders” in the project.

If you are developing an information system to be used internally within your company, you may include people with end-user experience and business domain expertise in your development team. Very often you will start the discussions at a business-model level rather than at a system level. If you are developing a product to be sold to a marketplace, you may make extensive use of your marketing people to better understand the needs of customers in that market.

Techniques for eliciting requirements include interviews, brainstorming, conceptual prototyping, questionnaires, and competitive analysis. The result of requirements elicitation is a list of requests or needs that are described textually and graphically, and that have been given priority relative to one another.

Key Skill 3: Defining the System

To define the system means to translate and organize the understanding of stakeholder needs into a meaningful description* of the system to be built. Early in system definition, decisions are made on what constitutes a requirement, documentation format, language formality, degree of requirements, request priority and estimated effort, technical and management risks, and scope. Part of this activity may include early prototypes and design models directly related to the most important stakeholder requests.

The outcome of system definition is a description of the system that is both natural language and graphical. Some suggested formats for the description are provided in later sections.

* We use the word “description” rather than “document” to avoid the perceived limitation inherent in the common use of the latter. A description may be a written document, electronic file, a picture, or any other representation meant to communicate system requirements short of the system itself.

Principle 55
Write Natural Language Before A More Formal Model

If you write the formal model first, the tendency will be to write natural language that describes the model instead of the solution system. Consider the following examples:

TO MAKE A A DIAL TONE. THE USER SHOULD DIAL A “9”. THE SYSTEM SHALL RESPOND WITH A DISTINCTIVE DIAL TONE...

THE SYSTEM CONSISTS OF FOUR STATES: IDLE, DIAL TONE, DISTINCTIVE DIAL TONE, AND CONNECTED. TO GET FROM THE IDLE STATE TO THE DIAL TONE STATE, LIFT THE PHONE. TO GET FROM THE DIAL TONE STATE TO THE DISTINCTIVE DIAL TONE STATE, DIAL A “9.”

Note that in the latter example, the text does not help the reader at all.

- Alan M. Davis, 201 Principles of Software Development, 1995

Key Skill 4: Managing the Scope of the Project

The scope of a project is defined by its requirements. Managing project scope to fit the available resources (time, people, and money) is key to managing successful projects. Managing scope is a continuous activity that requires iterative or incremental development, which breaks project scope into smaller more manageable pieces.

Using requirement attributes, such as priority, effort, and risk, as the basis for negotiating the inclusion of a requirement is a particular useful technique for managing scope. Focusing on the attributes rather than the requirements themselves helps desensitize negotiations that are otherwise contentious.

Key Skill 5: Refining the System Definition

With an agreed-upon high-level system definition and a fairly well understood initial scope, it is both possible and economical to invest resources in more refined system definitions. Refining the system definition includes two key considerations: developing more detailed descriptions of the high-level system definition, and verifying that the system will comply with stakeholder needs and behave as described.

The descriptions are often the critical reference materials for project teams. Descriptions are best done with the audience in mind. A common mistake is to represent what is complex to build with a complex definition, particularly when the audience may be unable or unwilling to invest the critical thinking necessary to gain agreement. This leads to difficulties in explaining the purpose of the system to people both inside and outside the project team. Instead, you may discover the need to produce different kinds of descriptions for different audiences. This paper includes suggested formats for detailed natural language, formal text, and graphical descriptions. Once the description format is established, refinement continues throughout the project lifecycle.

Key Skill 6: Managing Changing Requirements

No matter how carefully you define your requirements, they will change. In fact, some requirement change is desirable! It means that your team is engaging your stakeholders. Accommodating changing requirements is a measure of your team’s stakeholder sensitivity and operational flexibility – team attributes that contribute to successful projects. Change is not the enemy, **unmanaged** change is.

A changed requirement means that more or less time has to be spent on implementing a particular feature, and a change to one requirement may have an impact on other requirements. Managing requirement change includes activities such as establishing a baseline, keeping track of the history of each requirement, determining which dependencies are important to trace, establishing traceable relationships between related items, and maintaining version control. As Figure 3 illustrates, it is also important to establish a change control or approval process,

requiring all proposed changes to be reviewed by designated team members. Sometimes this single channel of change control is called a Change Control Board (CCB).

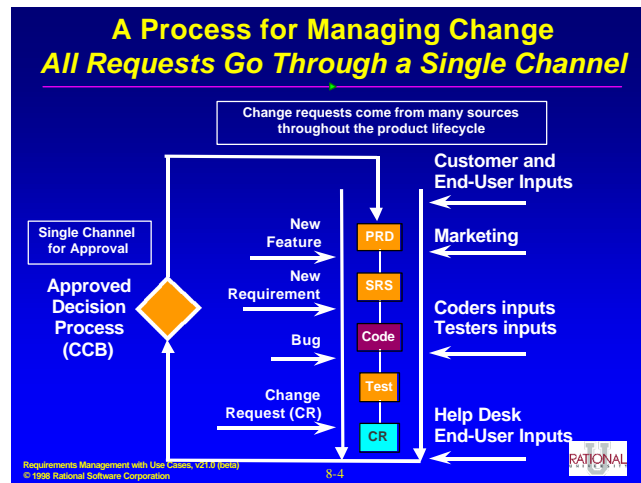


Figure 3: Process for Managing Change

Important Requirements Concepts

To apply requirements management skills to a project, certain requirements management concepts are useful for everyone on the project to understand. They include:

Requirement Type

The larger and more intricate the system, the more types of requirements appear. A requirement type is simply a class of requirements. By identifying types of requirements, teams can organize large numbers of requirements into meaningful and more manageable groups. Establishing different types of requirements in a project helps team members classify requests for changes and communicate more clearly.

Usually, one type of requirement can be broken down, or decomposed, into other types. Business rules and vision statements can be types of high-level requirements from which teams derive user needs, features, and product requirement types. Use cases and other forms of modeling drive **design requirements** that can be decomposed to **software requirements** and represented in analysis and design models. **Test requirements** are derived from the software requirements and decompose to specific test procedures. When there are hundreds, thousands, or even tens of thousands of instances of requirements in a given project, classifying requirements into types makes the project more manageable.

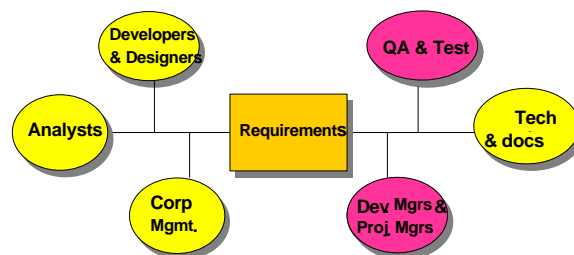


Figure 4: Cross Functional Teams

Cross-Functional Teams

Unlike other processes, such as testing or application modeling, which can be managed within a single business group, requirements management should involve everyone who can contribute their expertise to the development process. It should include people who represent the customer and the business expectations. Development managers, product managers, analysts, systems engineers, and even customers should participate. Requirements teams should also include those who create the system solution – engineers, architects, designers, programmers, technical writers, and other technical contributors. Testers and other QA personnel should be counted as important team members.

Often, the responsibility for authoring and maintaining a requirement type can be allocated by functional area, further contributing to better large project management. The cross-functional nature of requirements management is one of the more challenging aspects of the discipline.

Traceability

As implied in the description of requirement types, no single expression of a requirement stands alone. Stakeholder requests are related to the product features proposed to meet them. Product features are related to individual requirements that specify the features in terms of functional and non-functional behavior. Test cases are related to the requirements they verify and validate. Requirements may be dependent on other requirements or they may be mutually exclusive. In order for teams to determine the impact of changes and feel confident that the system conforms to expectations, these traceability relationships must be understood, documented, and maintained. While traceability is one of the most difficult concepts to implement in requirements management, it is essential to accommodating change. Establishing clear requirement types and incorporating cross-functional participation can make traceability easier to implement and maintain.

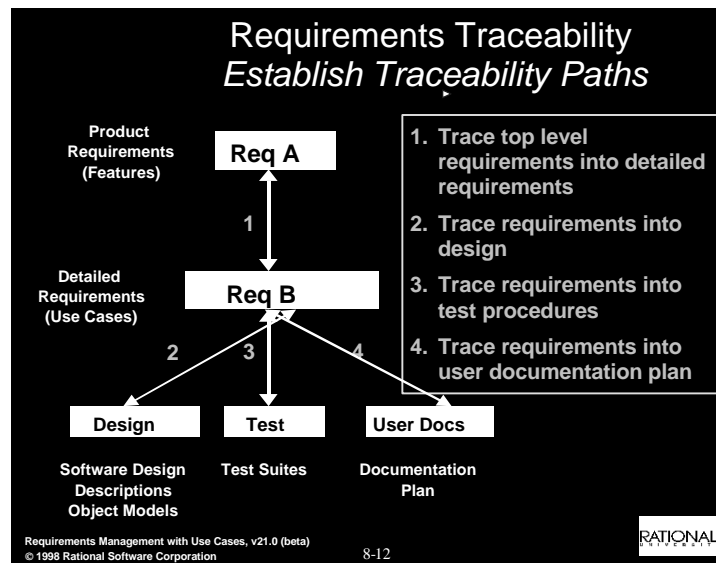


Figure 5: Requirements Traceability Established

Multi-Dimensional Attributes

Each type of requirement has attributes, and each individual requirement has different attribute values. For example, requirements may be assigned priorities, identified by source and rationale, delegated to specific sub-teams within a functional area, given a degree-of-difficulty designation, or associated with a particular iteration of the system. To illustrate, Figure 6 displays attributes for a Feature Requirement Type from a Learning Project in Rational's

RequisitePro requirements management tool. As implied by the title of the screen, the requirement type and attributes for each type are defined for the entire project, ensuring usage consistency across the team.

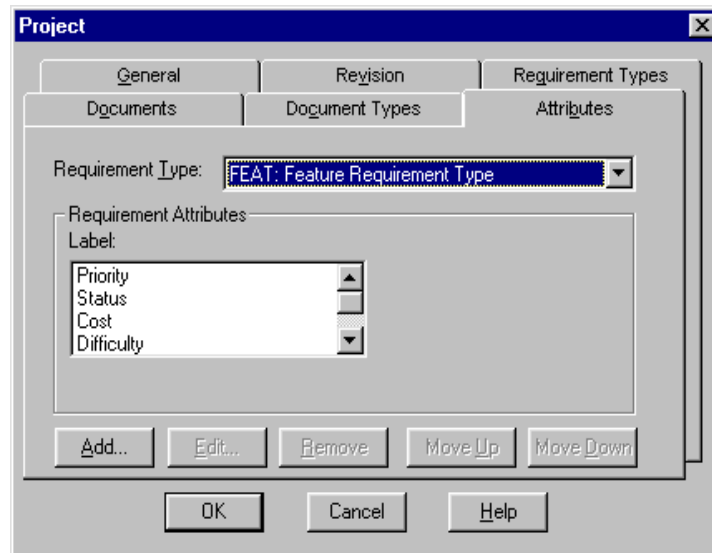


Figure 6: Feature Requirement Type from a Learning Project

In Figure 7, instances of feature requirements are displayed for a specific project in RequisitePro. Note that even without displaying the entire text for each requirement, we can learn a great deal about each requirement from its attribute values. In this case, its priority and difficulty – no doubt assigned by different members of the team – will help the team begin to scope the project to available resources and time, taking into account both stakeholder priorities and a very rough estimate of effort reflected in the difficulty attribute value. In more detailed types of requirements, the priority and effort attributes may have more specific values (e.g., estimated time, lines of code, etc.) with which to further refine scope. This multi-dimensional aspect of a requirement, compounded by different types of requirements – each with its own attributes – is essential to organizing large numbers of requirements and to managing the overall scope of the project.

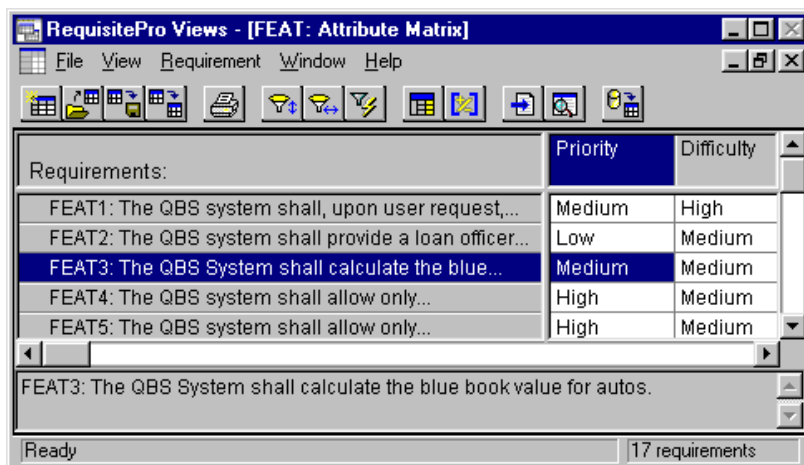


Figure 7: Feature Requirements for a Specific Project in RequisitePro

Change History

Both individual requirements and a collection of requirements have histories that become meaningful over time. Change is inevitable and desirable to keep pace with a changing environment and evolving technology. Recording the versions of project requirements enables team leaders to capture the reasons for changing the project, such as a new system release. Understanding that a collection of requirements may be associated with a particular version of software allows you to manage change incrementally, reducing risk and improving the probability of meeting milestones. As individual requirements evolve, it is important to understand their history: what changed, why, when, and even by whose authorization.

Putting Requirements Management to Work

Requirements management employs the key skills and concepts presented above to identify and resolve the problems successfully. To build a system that truly meets customers' needs, the project team must first define the problem to be solved by the system. Next, the team must identify stakeholders from whom business and user needs are elicited, described, and prioritized. From this set of high-level expectations or needs, a set of product or system features should be agreed upon.

Detailed software requirements should be written in such a form as can be understood by both the customers and the development team. We have found that using the language of the customer to describe these software requirements is most effective in gaining the understanding and agreement. These detailed software requirements are then used as input for the system design specifications as well as for test plans and procedures needed for implementation and validation. Software requirements should also drive the initial user documentation planning and design.

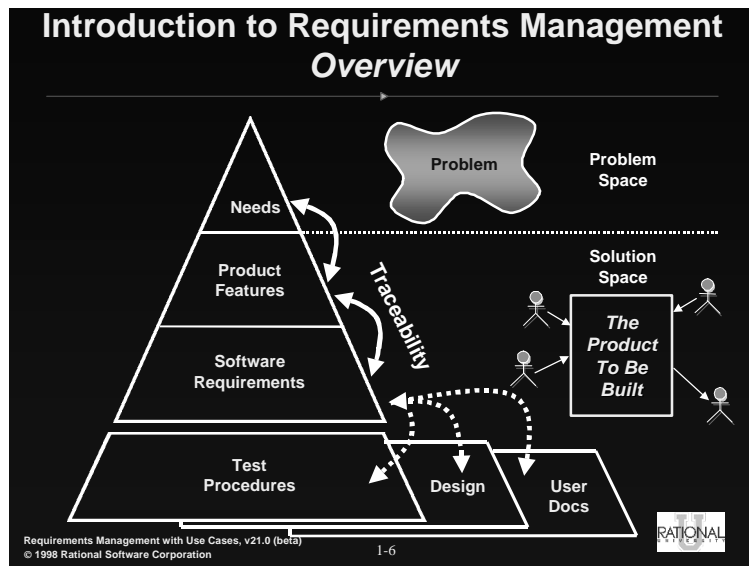


Figure 8: Requirements Management Overview

To facilitate this, the project team should:

- Agree on a common vocabulary.
- Develop a vision of the system that describes the problem to be solved by the system, as well as its primary features.
- Elicit stakeholders needs in at least five important areas: functionality, usability, reliability, performance and supportability.
- Determine what requirement types to use.

- Select attributes and values for each requirement type.
- Choose the formats in which requirements are described
- Identify team members who will author, contribute to or simply view one or more types of requirements.
- Decide what traceability is needed.
- Establish a procedure to purpose, review, and resolve changes to requirements.
- Create progress and status reports for team members and management.

These essential requirements management activities are independent of industry, development methodology, or requirements tools. They are also flexible, enabling effective requirements management in the most rigorous and the most rapid application development environments.

A Few Words about Documents

The decision to describe requirements in documents deserves some thought. On one hand, writing is a widely accepted form of communication and, for most people, a natural thing to do. On the other hand, the goal of the project is to produce a system, not documents.

Common sense and experience teach that the decision is not whether but *how* to document requirements. Document templates provide a consistent format for requirements management. Rational's RequisitePro offers these templates and the additional feature of linking requirements within a document to a database containing all project requirements. This unique feature allows requirements to be documented naturally while being made more accessible and manageable in a relational database.

Requirements Management Workflows

Requirements management can follow an infinite number of domain-specific paths. The following approach prescribes six detailed workflows that apply to each of the key requirements management skills but can be applied to any domain.*

Workflow: Problem Analysis

In the **Problem Analysis** workflow, the primary activity is *vision development*. Output from this activity is a *vision document* that identifies the high-level user or customer view of the system to be built. The vision expresses initial requirements as key features the system must possess in order to solve the most critical problems. The *system analyst* has the primary role in this workflow. The system analyst should have problem domain expertise and an understanding the problem, and should be able to describe a process that he or she believes will solve the problem. Active involvement from various project stakeholders is required.

To begin managing dependencies, features should be assigned attributes such as rationale, relative value or priority, and source of request. As the vision develops, the analyst identifies users and systems (the actors) of possible use

* The following workflow diagrams are from the Rational Unified Process Requirements Workflow. The workflows are expressed in terms of workers, activities and artifacts (input or output). The accompanying text in this paper describes each workflow briefly, in the hopes of stimulating your thoughts and interest in improving your requirements management process.

cases. Actors are the first element of the use-case model, which will define the system’s functional and non-functional technical requirements.

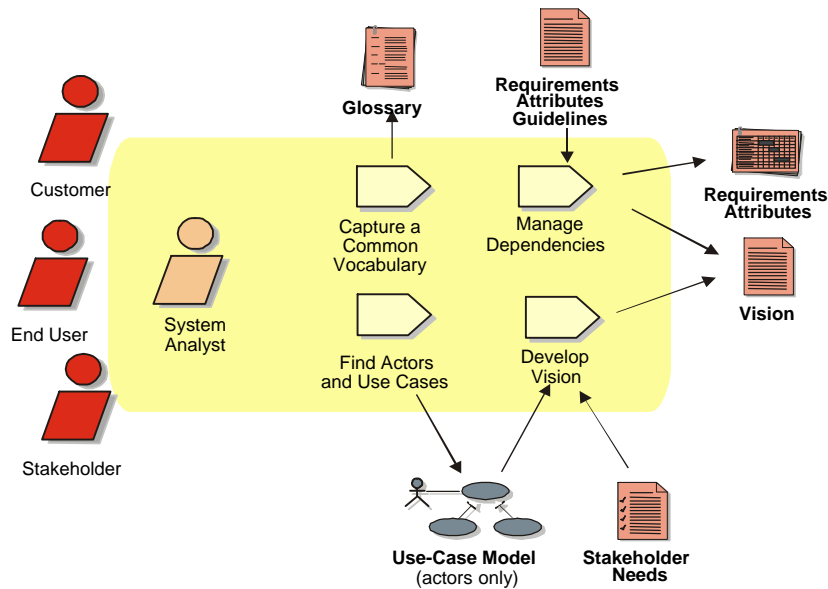


Figure 9: Problem Analysis

The Activities in Problem Analysis

Initiation: One or more stakeholders who perceive a problem will initiate the workflow.

One or more system analysts in a development team conduct a session to help the initial stakeholders describe the problem they want solved. The elements of the vision document are organized in the following table:

| | |
|------------------------------|---|
| Problem | Define the problem |
| Affected Stakeholders | List the stakeholders affected by the problem |
| Impact | Describe the impact of the problem |
| Successful Solution | List some key benefits of a successful solution |

The problem statements succinctly explain the purpose of the project. Problem analysis stimulates further investigation into all stakeholder needs and the initial business case including compelling benefits and roughly estimated costs. In parallel with defining problem statements, you should also compile a glossary by keeping track of commonly used terms and agreeing on their definitions.

Problem analysis also identifies the main system actors. Actors are users of the system or any other system that will exchange information with it. At this stage, problem analysis should briefly identify some obvious ways that the actors will interact with the system. Descriptions should be oriented towards business process rather than system

behavior. For example, a budgeting program may allow one type of actor to “Create departmental budget,” while another actor will be able to “Consolidate departmental budgets.” The system analyst may later break them into additional use cases that align more meaningfully with specific system behavior. For example, “Create departmental budget” could result in system use cases such as “Import spreadsheet information” and “Create budget views.”

Use-Case Model Introduction

Model Introduction

A use-case model consists of actors, use cases, and relations among them. *Actors* represent everything that must exchange information with the system, including what are typically called users. When an actor uses the system, the system performs a *use case*. A good use case is a sequence of transactions that yields a measurable result of value for an actor. The collection of use cases is the system’s complete functionality.

Jacobson I., Christerson M., Jonsson P., Overgaard G., Object-Oriented Software Engineering – A Use Case Driven Approach, Addison Wesley – ACM Press, 1992

The problem analysis session described above is often performed more than once, maybe with different stakeholders, and intermingled with internal development team sessions. The system analyst who conducted the meeting with the stakeholders will lead a session with members of the development team to envision a technical solution to the problems, derive features from the initial stakeholder inputs, and draft the vision description, the first definition of the system to be built. To facilitate understanding of the proposed solution among the initial stakeholders, the system analyst may use modeling tools or manual drawing techniques to complement the vision description.

The initiating stakeholders are consulted at multiple points to help refine the problem description and constrain the number and scope of possible solutions. Stakeholders and system analysts manage dependencies in this workflow by negotiating the priority of key features and gaining a general understanding of the resources and effort needed to develop them. While priority and effort/resource estimates inevitably change, managing dependencies early establishes an important pattern that continues throughout the development lifecycle. It is the essence of the scope management and an early predictor of project success.

After several drafts, the vision reaches a point when the team must decide whether to invest in additional requirements elicitation. By the same time, the business case approval process has been initiated separately. Although not addressed further in this paper, the business case describes:

- The context (product domain, market and scope),
- The technical approach,
- The management approach (schedule, risk, objective measure of success), and the financial forecast.

Workflow: Understanding Stakeholder Needs

If the initial vision justifies additional investment, the **Understanding Stakeholder Needs** workflow begins in earnest. The key activity is *eliciting stakeholder needs*. The primary outputs are collections of *prioritized stakeholder needs*, which enable refinement of the vision document, as well as a better understanding of the requirements attributes. Also, during this workflow you may start discussing the system in terms of its use cases and actors. Another important output is an updated glossary of terms to facilitate common vocabulary among team members.

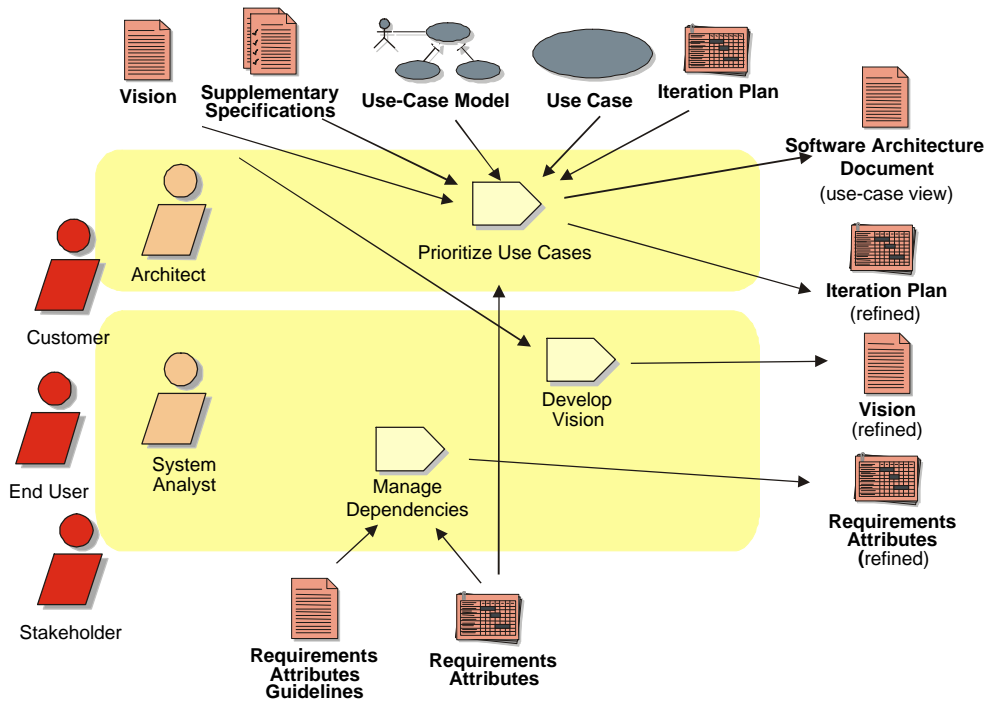


Figure 10: Understanding Stakeholders Needs

Activities in Understanding Stakeholders Needs

The system analyst and key stakeholders identify additional stakeholders and elicit their needs via interviews, workshops, storyboards, business process use cases, and other techniques. One or more system analysts facilitate these sessions. Requirements are among the most useful elicitation techniques. The process includes users, help-desk personnel, business owners, testers, and others who have a stake in the outcome of the proposed project. Stakeholder needs are often ambiguous, overlapping, and even extraneous. In addition to formal elicitation results, stakeholder needs may be expressed in well-formatted documents, defect and enhancement requests from databases, or e-mail and groupware threads. System analysts record, categorize, and prioritize stakeholder needs.

Understanding Stakeholder Needs: Where “Delighting Customers” Begins

Stakeholder needs are a type of proposed requirement captured as much as possible in the language and format of the submitting stakeholder. Unlike subsequent requirement types that are usually authored by process-educated and technically proficient project team members, stakeholder needs are often expressed poorly. They are duplicated or overlap. They can be expressed on anything from slips of paper to enhancement-request databases.

The analyst (or team representing the analyst role) must review them all, interpreting, grouping, perhaps retyping (without rewriting), and translating them into features in the vision description. Depending on the rigor applied in your development and the availability of tools, traceability between some or all stakeholder needs and features can be applied to help stakeholders understand how their needs were taken into account.

Demonstrating serious concern for eliciting and satisfying stakeholder needs by applying the Understanding Stakeholder Needs Workflow can be critical to establishing stakeholder confidence in your team’s abilities.

Based on a better understanding of stakeholders needs, the system analysts in the development team refine the vision document, paying special attention to the product position statement. In two or three sentences, this statement establishes the compelling value of the project. The statement should include intended users, the problems it solves,

the benefits it delivers, and the competitors it replaces. All team members should understand this project theme. System analysts also update the glossary to facilitate common understanding of terms.

Key stakeholders are consulted at multiple points to negotiate priority of new features derived from understanding stakeholder needs and gain a current understanding of resources and effort needed to develop them. As with problem analysis, managing dependencies in this workflow helps manage scope. It also establishes traceability between stakeholder needs and vision features, so stakeholders can be sure their inputs were considered.

Workflow: Defining the System

The **Problem Analysis** workflow and the **Understanding Stakeholder Needs** workflow create early iterations of key system definitions, including the vision document, a first outline to the use-case model, and the requirements attributes. The **Defining the System** workflow completes the description of the system-level requirements with the addition of new actors, use cases, and supplementary specifications.

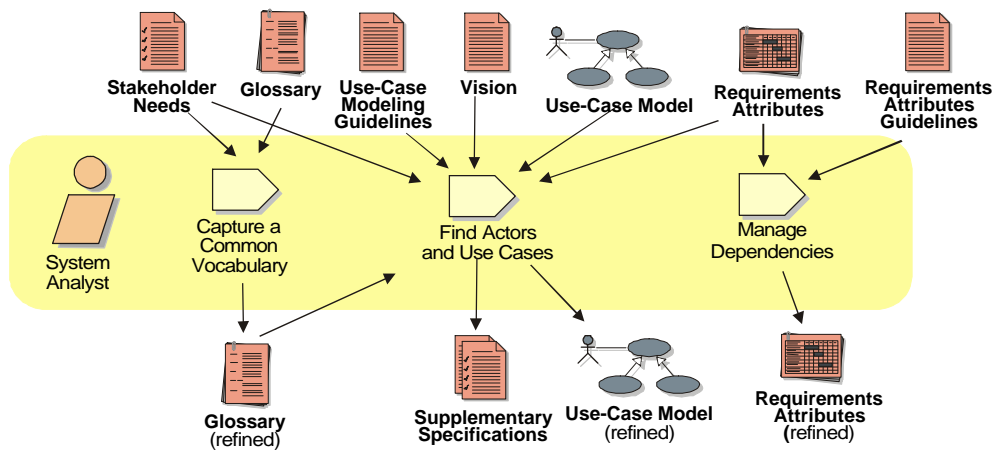


Figure 11: Defining the System

Activities in Defining the System

The glossary is updated to reflect current understanding about the terms used to describe features and the use-case model.

The system analyst uses the refined vision to derive and describe the use cases that elaborate on the system’s expected behavior. The use-case model serves as a contract between the customer, the users, and the system developers. It defines expectations for system developers and helps customers and users to validate that the system will meet these expectations.

The system analyst describes requirements that do not fit well in use cases in a supplementary specification. Usability, reliability, performance, and supportability requirements often end up here. It should be noted that many non-functional requirements of these types are specific to a single use case. It is better for use case authors to place these requirements in the use case specification itself (see the **Refining the System** workflow), leaving the supplementary specification for global non-functional requirements.

In this workflow, the system analyst creates attributes for the supplementary requirements (such as priority and related use cases). In addition, the system analyst adds and updates attribute values for the initial and new use cases. Finally, the system analyst manages dependencies by tracing important user needs and critical features to related use cases and supplementary specifications.

Workflow: Managing Scope

Identifying most actors, use cases, and supplementary specifications allows the system analyst to apply priority, effort, cost, and risk values to requirements attributes more accurately. This better understanding also enables the architect to identify the architecturally significant use cases.

Experience teaches that the keys to managing scope successfully are the well-considered attribute values assigned to stakeholder needs, use cases, and supplementary specifications; and regular, open, and honest interaction with representative stakeholders.

The iteration plan, developed in parallel by project and development management, first appears in the **Managing Scope** workflow. Also known as a development plan, the iteration plan defines the number and frequency of iterations planned for the release. The highest risk elements within scope should be planned for early iterations.

Other important outputs from the **Managing Scope** workflow include the initial iteration of the software architecture document* and a revised vision that reflects analysts and key stakeholders' increased understanding of system functionality and project resources.

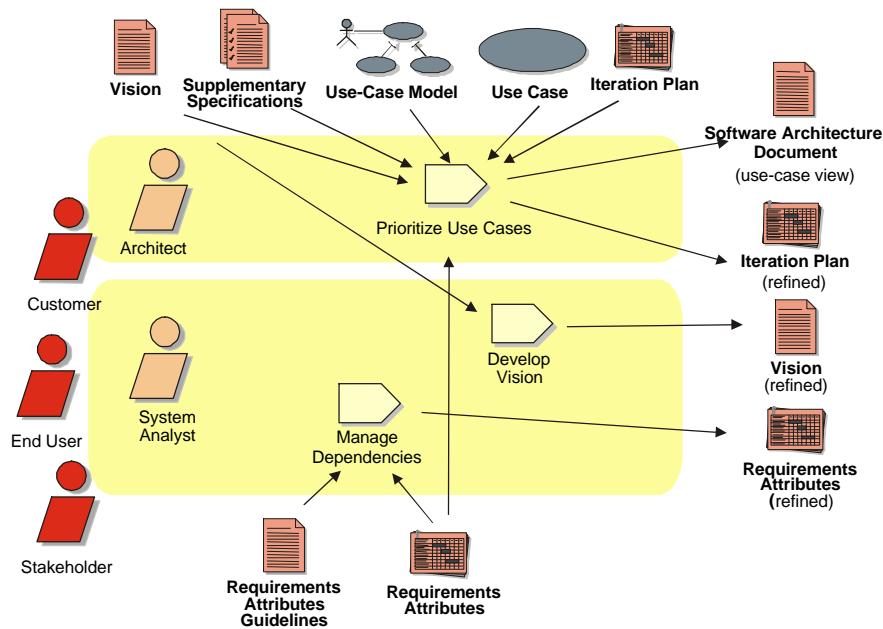


Figure 12: Managing Scope Workflow

Activities in Managing Scope

Architects prioritize use cases for their risk coverage, architectural significance, and architectural coverage. While the system may be defined with many use-case and supplementary specification requirements, only a subset of use cases are usually critical to good system architecture. With prioritized use cases, architects refine the iteration or development plan and model a use-case view of the system architecture in tools such as Rational Rose.

* Like the business case earlier and first issue of the iteration plan, the software architecture document is not an artifact of requirements management workflows, although it is related and is part of Rational Unified Process. It is not the subject of this paper.

System analysts manage dependencies by refining requirements attributes for features in the vision. They also refine requirements in use cases and supplementary specifications. System analysts ensure that appropriate traceability* exists for stakeholder needs, features, use-case requirements, and supplementary specification requirements.

System analysts negotiate revised project scope and vision with key stakeholders. This step is among the most important in the entire project. For the first time the **breadth** of knowledge about the proposed system is available to make serious commitments on requirements, project resources and delivery dates. At the same time, it must be understood that these requirements will change as the **depth** of knowledge increases. If dependencies have been managed in the previous three workflows, this step is much easier, and future changes will be easier.

Managing scope to match available resources is successful only if stakeholders and development team members view this step as a natural progression – not an ambush on users’ expectations or an attempt to blackmail the organization for more time and money. This workflow will need to be repeated at major milestones in the project to assess whether new insight into the system and its problems requires change to the scope. While committed requirements, budgets, and deadlines are hard to change, an in-depth understanding of prioritized use cases, supplementary specifications, and early system iterations inevitably lead to scope reconsideration.

Once again, it is critical that the project team engages in habitual scope management before reaching the refinement stage. Representative stakeholders must understand and trust that their priorities and interests are taken seriously during increasingly difficult scope negotiations. By the time system requirements are refined, only important requirements remain to be negotiated or modified. Unless effective scope management habits have been established, your project may be doomed as a “death march” – a hopelessly over-scoped project moving inexorably towards delays and cost overruns.

Workflow: Refining the System

The **Refining the System** workflow assumes that system-level use cases have been outlined and actors have been described, at least briefly. Through managing the project scope, the features in the vision have been re-prioritized and are now believed to be achievable by fairly firm budgets and dates. The output of this workflow is a more in-depth understanding of system functionality expressed in **detailed use cases, revised supplementary specifications, and early iterations of the system** itself*.

* The term “appropriate traceability” is deliberate. See the inset text on Traceability later in this paper.
* Obviously, not all systems will have user interfaces and not all early iterations will include GUI elements. We use them here only as an example of an early iteration. Other examples include prototypes, models, and storyboards.

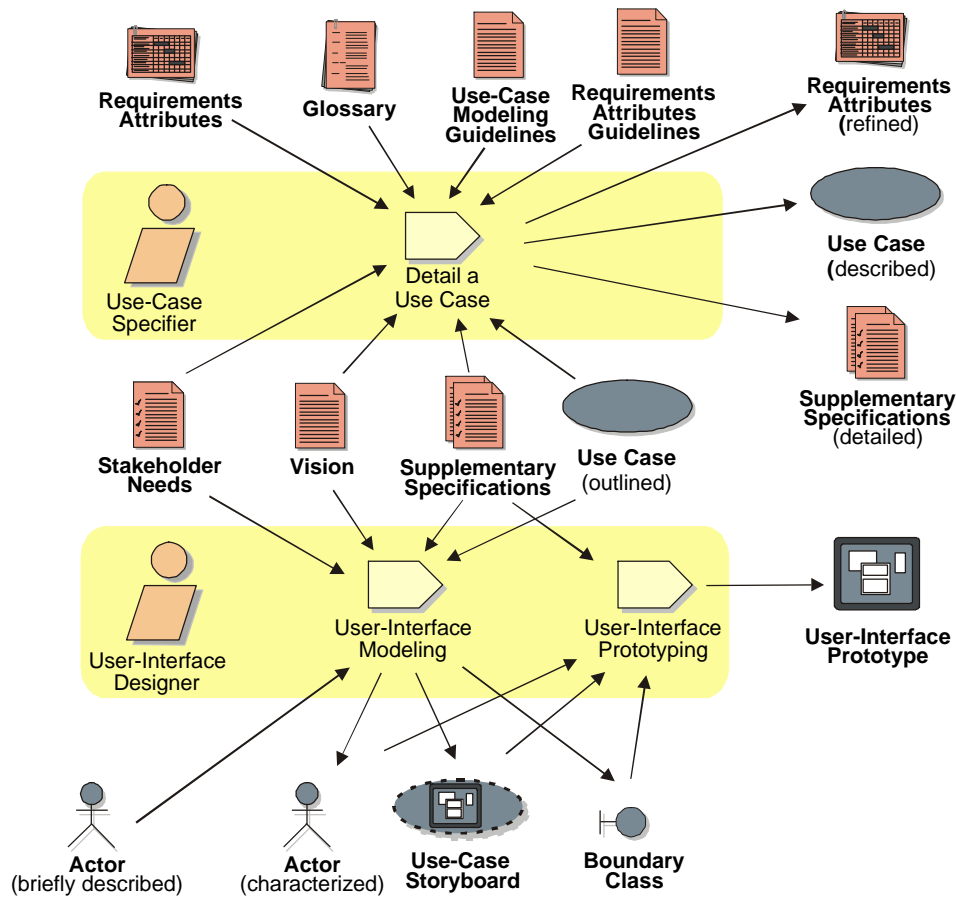


Figure 13: Refining the System Workflow

Activities in Refining the System

The **use-case specifier** details the definition of the flow of events, pre- and post-conditions, and other textual properties of each use case. To minimize the effort and enhance the readability, it is advisable to use a standard document format or a use case specification template, to capture textual information about each use case. Creating well-thought-out use case specifications is critical to the quality of the system. This specification development requires a thorough understanding of the stakeholder needs and features related to the use case. It is desirable to have several members of the project team (such as software engineers) participate in creating the use cases.

In parallel, the use case specifier revises the supplementary specification with additional requirements that are note specific to the use case.

The **user-interface designer** models and prototypes the user interface of the system. This work is highly correlated to the evolution of the use cases.

The use-case specifier and the system analyst revise the effort, cost, risk, and other attribute values for each requirement that is understood better.

The result of this system refinement is submitted to another round of the **Managing Scope** workflow. Once you know more about the system, you may want to change the priorities.

Workflow: Managing Requirement Change

Like the **Managing Scope** workflow, the **Managing Requirements Change** workflow should be applied continuously. The output of this workflow can cause modification to every artifact, which requires effective communication with all project team members and stakeholders.

In this workflow we **introduce additional artifacts** that are affected by requirements workflows. Changes to requirements naturally affect the system models that represent them in the analysis and design workflows. Requirement changes also affect tests created to validate the proper implementation of the requirements*. Traceability relationships identified in the process of managing dependencies are the keys to understanding these impacts.

Traceability

Much is made of traceability in the requirements field. Many promote the virtue of tracing individual customer requirements to each related specification, test, model element, and ultimately source code files. Certainly, some traceability is the key to successful requirements change management.

Be forewarned, however, that all forms of traceability require an investment to set up and maintain during the life of the project. Like all investments, traceability has diminishing points of return depending on your specific situation. This paper emphasizes the value of tracing between types of requirements. This is a good place to start and can be automated by tools such as Rational’s RequisitePro. We believe you will find some level of requirements traceability to be a good investment

Another important concept for Managing Requirements Change Workflow is **requirement history tracking**. By capturing the nature and rationale of requirements changes, reviewers (anyone on the software project team whose work is affected by the change) receive the information needed to respond to the change properly.

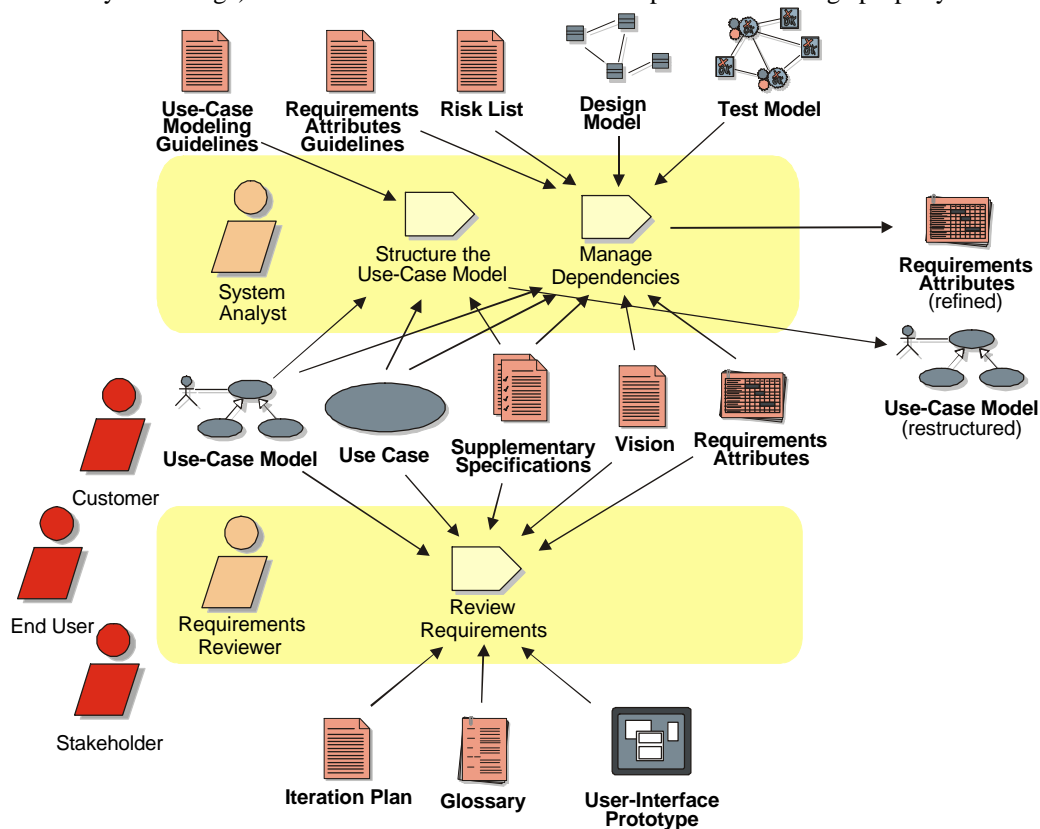


Figure 14 : Managing Requirements Change Workflow

* As in earlier examples, these artifacts are part of the Rational Unified Process but are not the subjects of this paper.

Activities in Managing Changing Requirements

Changes to requirements are initiated by any stakeholder or project team member for an infinite number of reasons.

The system analyst initiates a review activity, assimilating all change requests, and classifies them as:

- defects in implementation that do not affect requirements,
- modifications to existing requirements of some type, and
- new stakeholder needs or enhancement requests.

Once classified, the proposed changes to requirements are assigned attributes and values as described in other requirements workflows.

In reviewing changes, the system analyst presents proposed prioritized requirement changes to a **change control board** comprised of representative stakeholders and project team members. Scope modifications that exceed resources should be rejected or elevated to stakeholder representatives that are empowered to approve required changes to date and budget commitments.

The change control board approves or rejects changes to requirements.

The system analyst communicates requirements changes to requirements authors or makes changes directly to requirements in the vision, use cases, or supplementary specification documents.

The **requirements reviewers** (developers, testers, managers, and other project team members) evaluate the impact of changes to requirements on their work by reviewing requirement history. Finally, they implement the change and make appropriate changes to related requirements for which they have authority.

Summary

The need to manage requirements is not new. So, what makes the preceding information worth considering now?

First, if your projects are not regularly satisfying customers, meeting deadlines, and staying within budget, you have reason to reconsider your development approach. If in doing so, you determine that requirements-related problems are undermining your development efforts, you have reason to consider better requirements-management practices.

Second, the requirements management practices summarized in this paper embody the collective experience of thousands, and are the well-considered opinions of a number of individuals who have spent years working with customers in the field of requirements management.

The chronic, pervasive problems of requirements management are solvable. And that, ultimately, may be the best reason to start practicing excellence in requirements management today.

Rational

unifying **software** teams

Corporate Headquarters
18880 Homestead Road
Cupertino, CA 95014
Toll-free: 800-728-1212
Tel: 408-863-9900
Fax: 408-863-4120
E-mail: info@rational.com
Web: www.rational.com

For International Offices: www.rational.com/corpinfo/worldwide/location.jttml

Rational, the Rational logo, and Rational Unified Process are trademarks or registered trademarks of Rational Software Corporation in the United States and in other countries. All other names are used for identification purposes only and are trademarks or registered trademarks of their respective companies. ALL RIGHTS RESERVED. Made in the USA.

© Copyright 1999 by Rational Software Corporation.



How to Implement an Effective Requirements Management Process

Dave Locke, Director of Requirements Management Products,
Rational Software

High Cost of Requirements Errors

- ◆ Requirements errors are the most expensive errors
- ◆ Requirements errors are the most common errors
- ◆ Rework consumes 40 - 50% of project budget
- ◆ Requirements errors consume the majority (>70%) of rework costs
- ◆ Requirements errors consume 30 - 40% of the total project budget!

Rational
unifying software teams

Rational
unifying software teams

Tips and Techniques

- ◆ The Methods of Managing Requirements...
- ◆ To find, document, organize and track requirements, you must develop five basic skill areas:

Problem Analysis

Understanding the true business need

Elicitation

Techniques for discovering stakeholder requirements

System Definition

Techniques in problem analysis and documentation

Scope Management

Techniques to "objectify" the effort to match resources to project scope

Change Management

Mechanisms to accommodate change

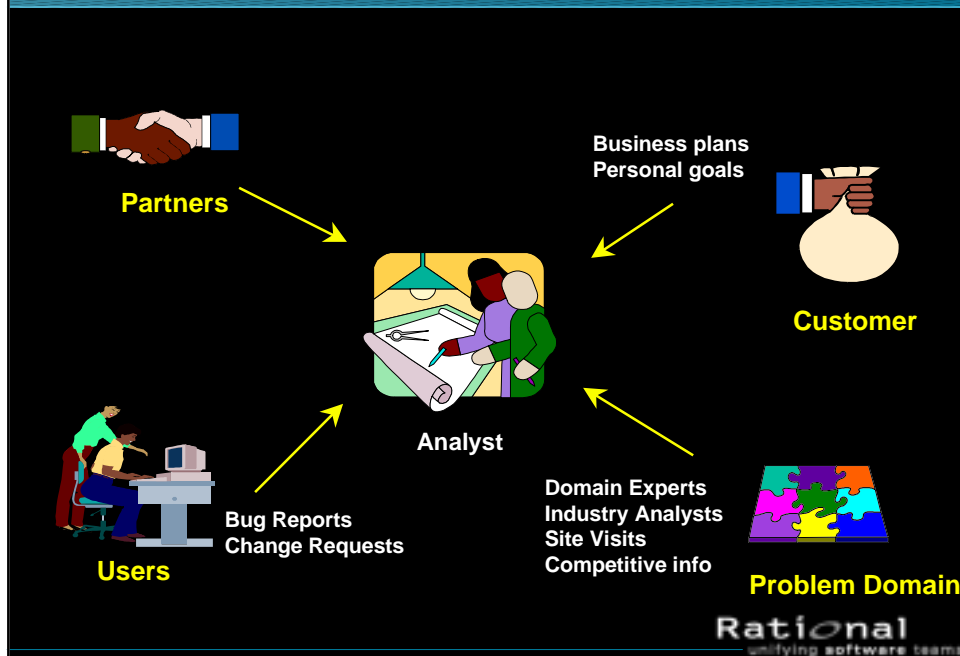
Rational
unifying software teams

Steps in Problem Analysis

- ◆ Step 1 - Gain agreement on the problem being solved
- ◆ Step 2 - Build a common vocabulary
- ◆ Step 3 - Identify the stakeholders (they are materially affected by outcome of the system)
- ◆ Step 4 - Define the system boundaries
- ◆ Step 5 - Identify constraints to be imposed on the system

Rational
unifying software teams

Typical Sources of Requirements



Several Elicitation Techniques

- Interviews
- Questionnaires
- Requirements Workshops
- Brainstorming & Idea Reduction
- Storyboarding
- Role Playing
- Prototyping
- Use Cases (Scenarios)

Interviews - The Context-Free Question

- **The context-free question is a high level, abstract question that can be posed early in a project to obtain information about global properties of the user's problem and potential solutions**
- **Context-free questions:**
 - Are always appropriate
 - Help you understand stakeholder perspectives
 - Are not biased with solutions knowledge

Gause & Weinberg, 1989

Rational
unifying software teams

Types of Context-Free Questions

User

- Who is the customer?
- Who is the user?
- Are their needs different?
- What are their backgrounds, capabilities, environments?

Process

- What is the reason for wanting to solve this problem?
- What is the value of a successful solution?
- How do you solve the problem now?

Product

- What business problems could this product create?
- What environment will the product be used in?
- What are your expectations for usability, reliability, performance?

Meta-Questions

- Am I asking too many questions?
- Do my questions seem relevant?
- Are you the right person to answer these questions?
- Are your answers requirements?
- Can I ask more questions later?
- Is there anything else I should be asking you?

Gause & Weinberg, 1989

Rational
unifying software teams

Rational
unifying software teams

Prototypes

Throw away

- Validates technological feasibility; exposes potential risks
- Throw away **everything** except knowledge gained

Evolutionary

- Demonstrates a proposed solution
- Throw away some (save core technologies)

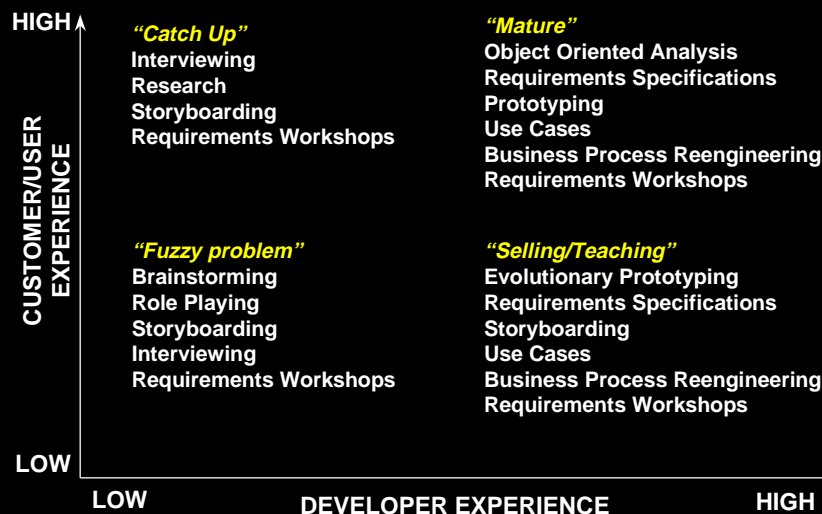
Operational Prototype

- Final form, function and fit, as well as technology
- Throw away **as little as possible**

Davis '95

Rational
unifying software teams

Applicability of Elicitation Techniques



Adapted from Alan Davis

Rational
unifying software teams

Learn to Like Your Users!

Users tend to be inarticulate, inconsistent, unavailable,
non-technical

BUT

We must learn to understand our users

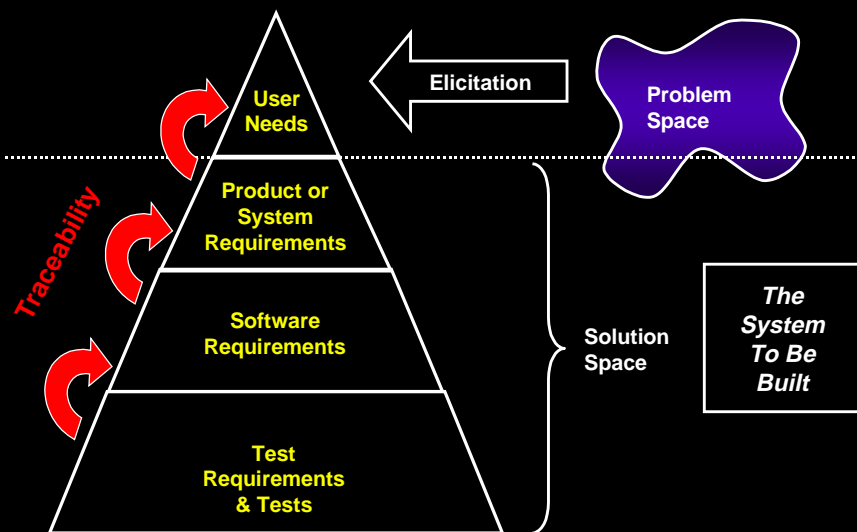
BECAUSE

If we don't get the **USERS'** requirements **RIGHT**,
we can't deliver **QUALITY** systems.

The users are not the enemy!

Rational
unifying software teams

The Requirements Management View



Rational
unifying software teams

FURPS

| | | |
|-----------------------|--|---|
| Functionality | Feature Set Capabilities | Generality Security |
| Usability | Human Factors Aesthetics | Consistency Documentation |
| Reliability | Frequency/Severity of Failure Recoverability | Predictability Accuracy MTBF |
| Performance | Speed Efficiency Resource Usage | Throughput Response Time |
| Supportability | Testability Extensibility Adaptability Maintainability Compatibility | Configurability Serviceability Installability Localizability Robustness |

Rational
unifying software teams

What vs. How

Is it a requirement or a design?

- ◆ A requirement should allow more than one design option. A design is a choice among options.
- ◆ A requirement that leaves no options is a *design constraint*
 - Should be distinguished from a requirement
 - Sources of each should be identified
 - The rationale for each should be documented

Rational
unifying software teams

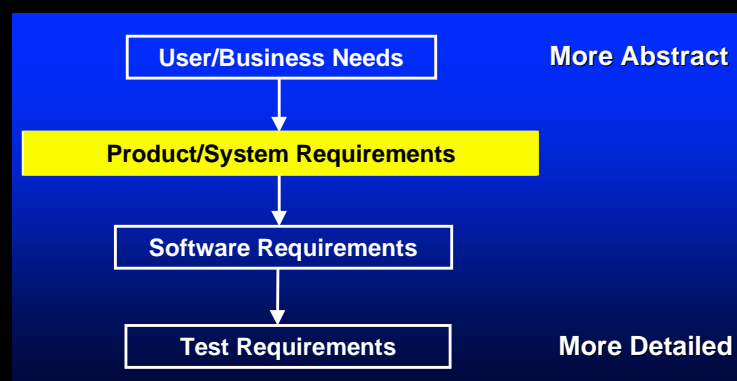
Standardize Your Document Format

- ◆ Leverages the work of others
- ◆ Documents appear familiar and unintimidating
- ◆ Documents are easier to write
 - Standard chapters and sections
 - Tips provided to help the writer
- ◆ Documents are easier to read
 - Know exactly where to look for information
- ◆ Ensure that important topics are covered
 - Mandatory sections act as checklist
- ◆ Ensure that things don't fall through the cracks

Use a standard look and feel for all documents of a specific type

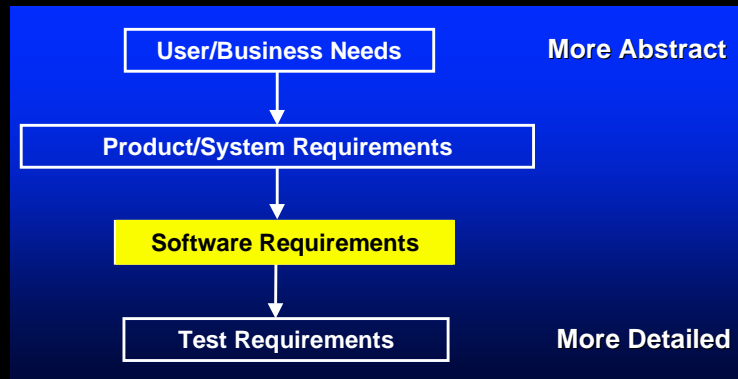
Rational
unifying software teams

Establish Requirement Hierarchies



Rational
unifying software teams

Establish Requirement Hierarchies



Rational
unifying software teams

Roles of the Software Requirements Specification

- ◆ Basis of communication between all parties
- ◆ Contractual agreement between parties
- ◆ The software manager's reference
- ◆ Input to design team
- ◆ Input to software test and quality assurance
- ◆ Controls evolution of system



Adapted from Alan Davis

Rational
unifying software teams

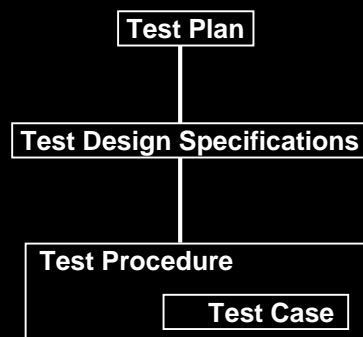
What's not in a SRS?

- ◆ Design information - How to accomplish the requirements
 - Design describes a sub-component of a system and/or its interfaces with other sub-components
- ◆ Project information
 - Schedules, verification and validation plans, configuration management plans, etc.
- ◆ How you'll know the requirements have been met
 - Test procedures
 - Acceptance procedures

Adapted from Alan Davis

Rational
unifying software teams

Software Test Documentation



Test Plan = Testing methodology, resources, etc.

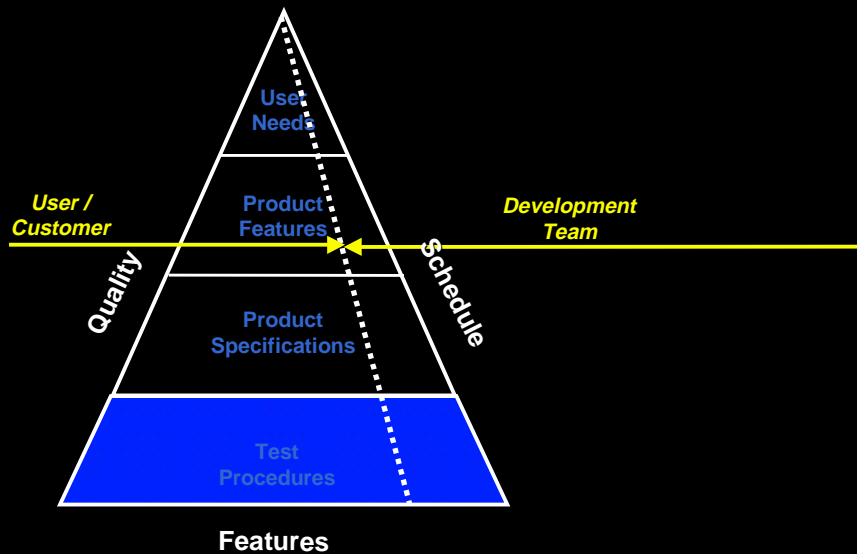
Test Design = Test structure, automation design

Test Procedure = Specific test path, step-by-step instructions

Test Case = Requirement to be verified

Rational
unifying software teams

Work vs. Resources

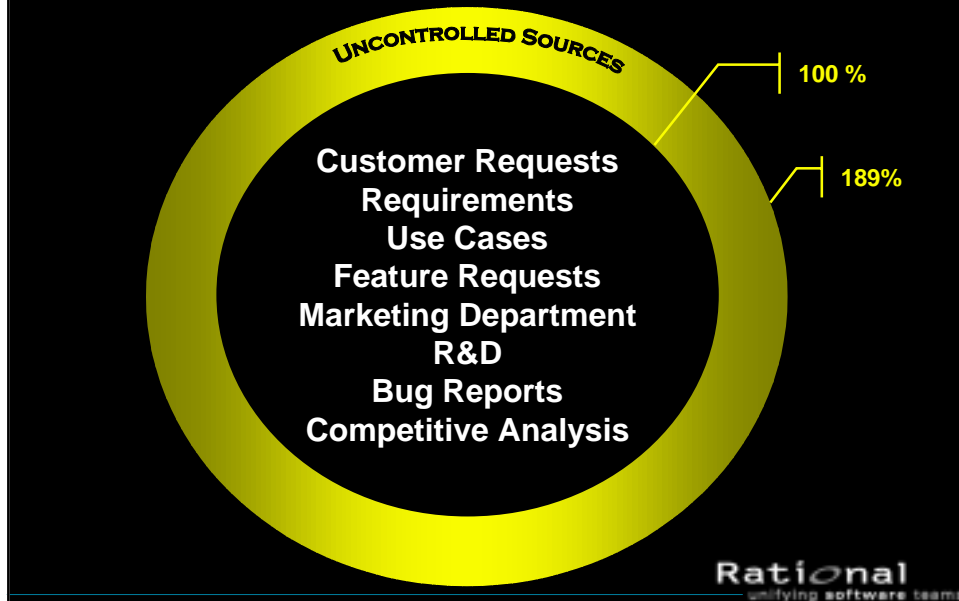


Establish a Requirements Management Process

- ◆ Management education
- ◆ Requirements training
- ◆ Infrastructure support:
 - People
 - Repository
 - Tools
 - Network / Email
- ◆ Process support
 - Requirement reviews
 - Change management policy

Rational
unifying software teams

Gain Control Early in the Process

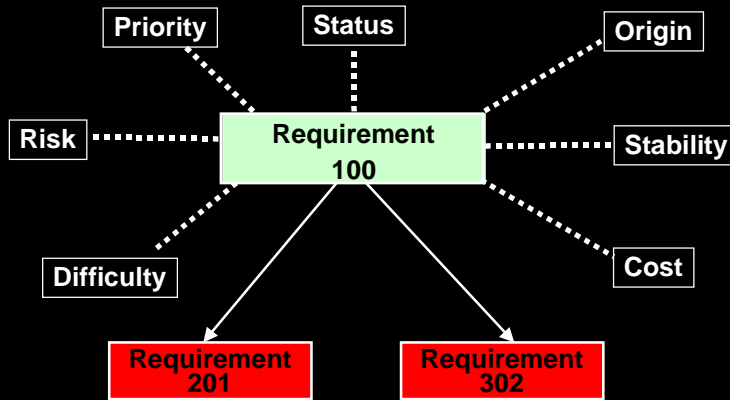


Communicate Requirements

- ◆ Document all requirements
- ◆ Itemize to an appropriate level of detail
- ◆ Make visible to all stakeholders
- ◆ Ensure that the requirements are reasonably stable
- ◆ Understand the rationale and benefits for each requirement
- ◆ Allow change - analyze impact of change before accepting requirement modification
- ◆ Maintain "living" documents that are easy to adapt to requirement changes
- ◆ Establish requirement relationships to indicate both dependencies and refinement

Rational
unifying software teams

Understand Your Requirement Attributes



Requirement ATTRIBUTES and RELATIONSHIPS
are a Rich Source of Management Information

Rational
unifying software teams

Objectively Decide What to Do

| Requirements | Priority | Difficulty | Risk | Stability | Action |
|--|----------|------------|----------|-----------|--------|
| REQ1: Save and restore sort and filter criteria. | Med High | Low | Low | High | |
| REQ2: Ability to save a Requisite document as a Word document. | Med High | Low | Low | High | |
| REQ3: Ability to see deleted requirements in a view window. | Medium | Med High | Medium | Medium | |
| REQ4: Support for Currency datatype attributes. | Medium | Medium | Med Low | Medium | |
| REQ5: Support the "All" document type (provides an easy way to define common attributes across multiple document types). | Med High | Medium | Medium | Med High | |
| REQ6: Ability to select requirement in a view and GoTo in Word document. | Med High | Medium | Medium | Med High | |
| REQ7: Display a requirements attribute in the text of the requirements document. | Medium | Medium | Medium | Med High | |
| REQ8: New project wizard | Med High | High | Med High | Medium | |
| REQ9: Fast creation of a requirement (avoid the requirement dialog on creation). | Med High | Med Low | Med Low | High | |
| REQ10: Autosave of a project (project archive). | Medium | Med Low | Medium | Medium | |
| REQ11: Change one or more attributes for a selected set of requirements. | Medium | Med High | Medium | Medium | |
| REQ12: Ability to Clone a project's structure to allow users to easily create new projects from old. | High | Medium | Medium | Low | |
| REQ13: Performance enhancements for printing, requirement identification. | High | Med High | Medium | Med High | |
| REQ14: Windows95 Port. | High | Medium | High | High | |

Rational
unifying software teams

Use Requirements Traceability

Customer's Business Needs

drive

Customer Needs

which drive

User Needs

which demand

Product Features

that drive

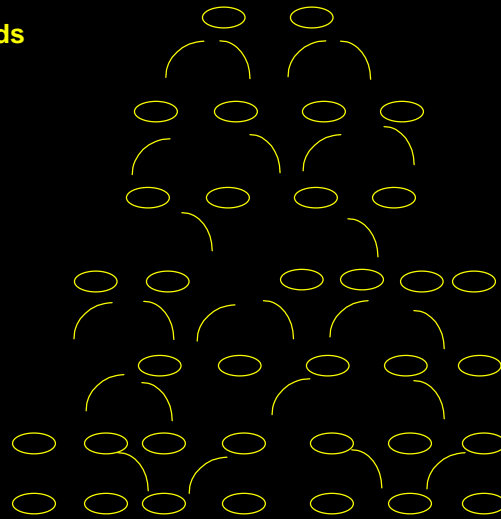
Software Requirements

that we, developers,

Implement

and

Test.



E. Magaziner '96

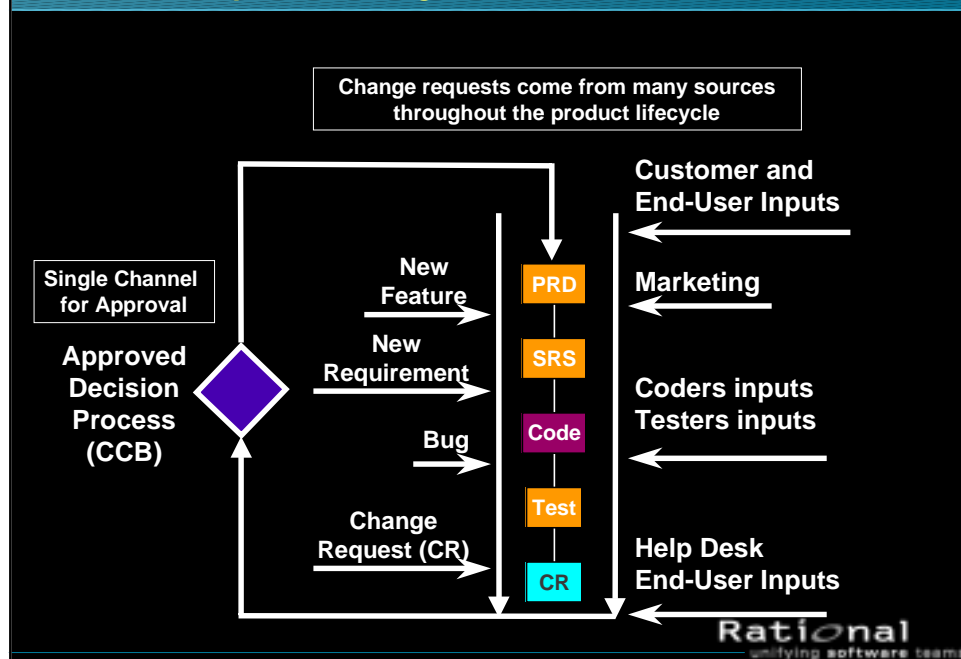
Rational
unifying software teams

What is Traceability?

- ◆ Requirements tracing is the linking of a requirement to other requirements and to other lifecycle elements
- ◆ The purpose of requirements tracing is to:
 - Verify that all requirements of the system are fulfilled by the implementation
 - Verify that the application does only what it was intended to do
 - Help manage change
- ◆ A proven technique for understanding the impact of changes
- ◆ A proven technique for assuring quality

Rational
unifying software teams

Route All Requests Through the RM Process



Watch-Out For Requirement Churn

- ◆ Requirement Churn = Excessive semantic changes to a requirement
- ◆ Visible if requirement's changes are tracked
- ◆ Sign of a poorly understood, ill-defined requirement
- ◆ Often the source of scope problems
- ◆ Measured by a requirement's *stability*

Do NOT work on unstable requirements!

Rational
unifying software teams

Use Cases and Requirements

- ◆ The next generation???
- ◆ Notation
- ◆ Benefits of use cases
- ◆ Use case example
- ◆ Use case documentation
- ◆ Use cases promote requirements reuse
- ◆ Use cases and documentation

Rational
unifying software teams

Use Case Notation

A use case is a sequence of actions a system performs that yields an observable result of value to a particular actor.

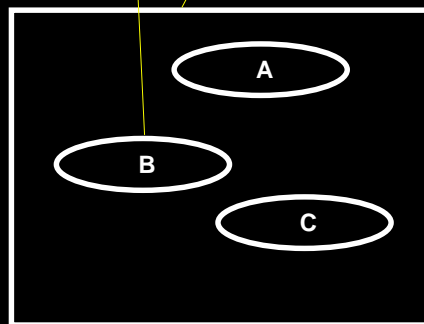


Actor

An actor is someone or something outside the system that interacts with the system.

Use Case

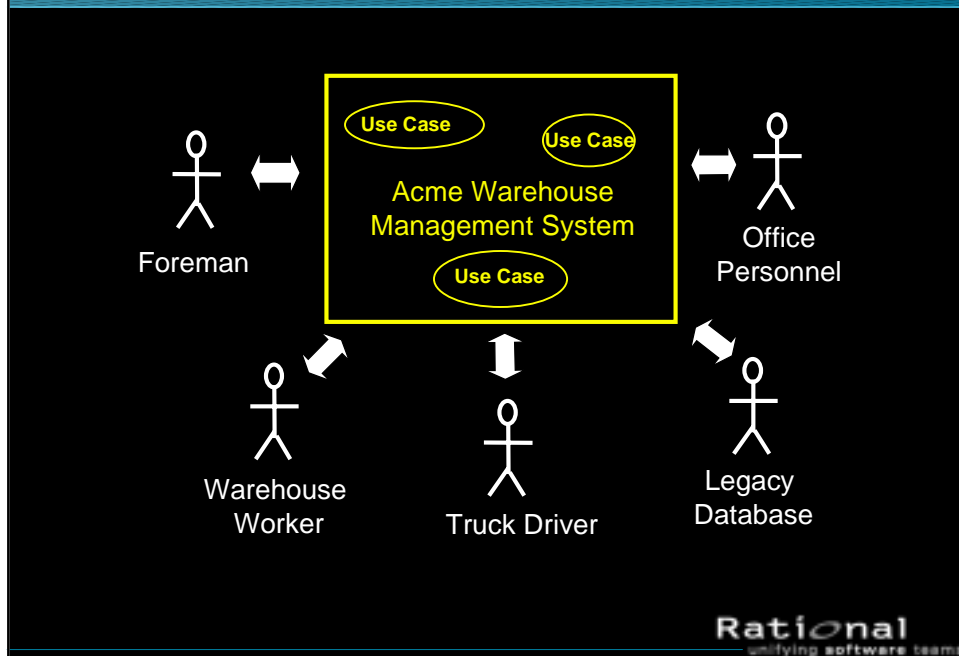
System Boundary



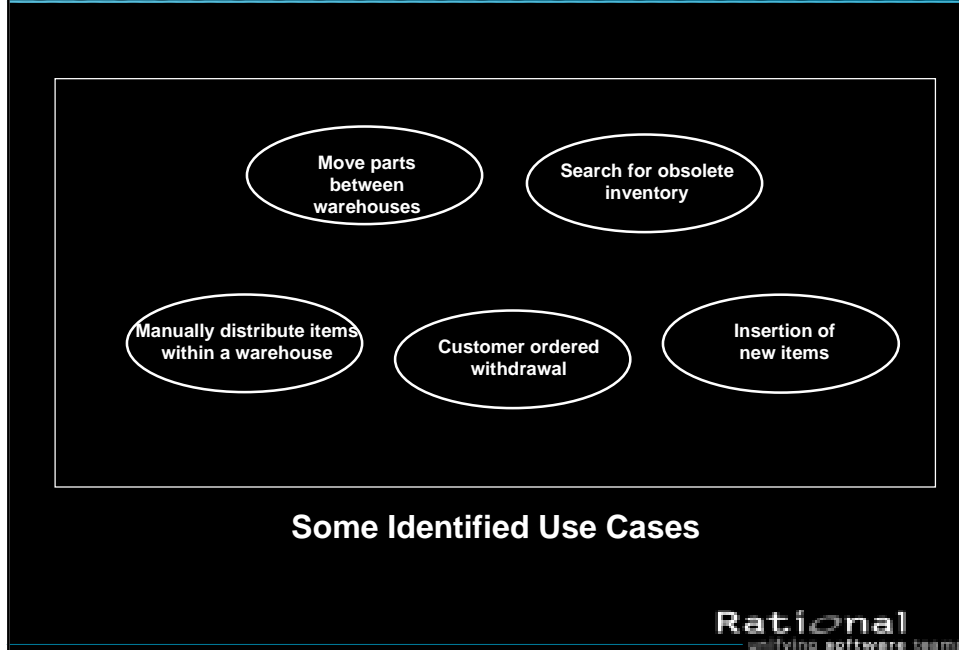
Jacobson '94

Rational
unifying software teams

Use Case Example - Warehouse Automation



Use Case Example - Identified Use Cases



Use Case Model Survey

- ◆ Move parts between warehouses
Used by warehouse workers to initiate the selection and transfer of warehouse parts from the central warehouse facility to satellite facilities.
- ◆ Insert new warehouse items
Used by warehouse personnel to add new warehouse items after they have been purchased and placed into item holding areas in area 10.
- ◆ Search for obsolete inventory
Used by the foreman to manually program the movement of a part from one area of the warehouse to another.
- ◆ Manually distribute items within a warehouse
Used by the foreman to manually program the movement of a part from one area of the warehouse to another.

Rational
unifying software teams

Use Case Specification

| | |
|----------------------------|---|
| Name: | Manually distribute items within a warehouse |
| Description: | Used by the foreman to manually program the movement of a part from one area of the warehouse to another. |
| Flow of Events: | The foreman gives a command for redistribution within a warehouse. The Window in Figure 3 is presented to the foreman. The items can be ordered a number of ways. This is selected with the ORDER menu item. Choices are: Alphabetical, Index, Storing In the "From Place" table, the foreman can view either all Places within the current |
| Alternative Flow A: | If the foreman is not authorized for manual distribution ... |

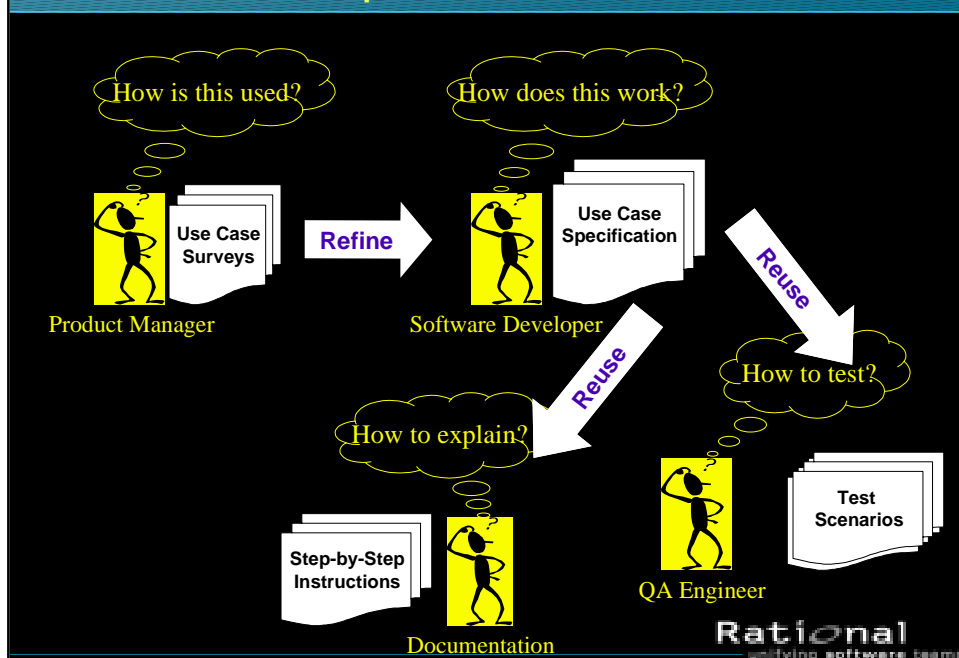
Rational
unifying software teams

Benefits of Use Cases

- ◆ Provide context around requirements by expressing sequences of events
- ◆ Most efficient communication mechanism with end users and domain experts
- ◆ Identify users and system boundaries
- ◆ Identify system interfaces
- ◆ Help us concentrate on the WHAT (rather than the HOW)
- ◆ Reusable in test and user documentation
- ◆ Good fit for all (OO or not) design methods

Rational
unifying software teams

Use Cases Promote Requirement Reuse



Rational
unifying software teams

Summary

Requirements and requirements management are
GOOD!

Not adequately addressing requirements
and requirements management is
BAD!

- ◆ There are a variety of useful tips and techniques at your disposal

Beware of the Dark Side...

Rational
unifying software teams

Modelling Dynamic Behaviour Based on Use Cases

Peter Fröhlich and Johannes Link¹

ABB Corporate Research

Speyerer Straße

Heidelberg, Germany

Email: peter.froehlich@decrc.mail.abb.com,

johannes.link@andrena.de

Keywords

UML, Requirements Analysis, State Machine, Use Case, Test Automation.

Abstract

In most object-oriented software development methods use cases are in the centre of the requirements analysis phase. We argue that a controlled process requires a formal mapping of use cases to a representation with stricter semantics. In this paper we describe a method to transform use cases into state machines. Our method considers all elements usually specified in a use case and integrates all its scenarios into a single state chart.

1 The role of use cases in the software development process

Use Cases [6] are a popular formalism for capturing functional requirements and business requirements. Use cases are a good means to communicate with a customer. They are the unit of work in incremental object-oriented software processes like the Rational Unified Process. Furthermore, they are a good basis for systematic testing. In [8] Rumbaugh et al. define a use case as "the specification of sequences of actions, including variant sequences and error sequences that a system, subsystem or class can perform by interacting with outside actors". While the advantages of use cases for requirements engineering are widely accepted, the impact of use cases on software design is less clear. The UML meta model [11] offers three different notations for designing dynamic system/subsystem/class behaviour based on use cases, as discussed in [12]:

- *Activity diagrams* interpret use cases as branching processes.
- *Interaction diagrams* (sequence diagrams and collaboration diagrams) can be used to formalize single scenarios contained in use cases.
- *State diagrams* specify the behaviour of a system/subsystem/class in reaction to events from actors. In contrast to interaction diagrams they visualize multiple scenarios, e.g. the hierarchy of scenarios described by a use case.

1. Now employed at Andrena Objects GmbH

2 Previous Work

2.1 Use Cases to Activity Diagrams

Activity diagrams have recently been added to UML and act as a variant of state diagrams. While state diagrams describe the lifecycle of an object using state transitions caused by operation invocations, activity diagrams show workflows, i.e. here the state transitions are caused by the termination of an action [8]. Some authors [13] recommend the use of activity diagrams for the formalization of use cases. Moreover, activity diagrams are a good means to communicate with the customer, especially if the customer is used to flow diagrams.

However, the disadvantages of using activity diagrams for formalizing use cases are considerable. First, state diagrams correspond directly to the object-oriented paradigm (states are caused by events, i.e. operations on a class), whereas activity diagrams model the control flow of a program. Thus they provide the same means for creating a spaghetti control-structure as do flow diagrams. Second, the distinction between normal and abnormal behaviour, which is one of the benefits of use case analysis is lost when using activity diagrams. The user has to consider all the choices in the control structure without being guided through the intended behaviour of the system first.

2.2 Use Cases to Interaction Diagrams

In OMT [7] Rumbaugh proposes to start dynamic modelling with scenarios in text form. These scenarios are then more formally expressed as sequences of events and shown in interaction diagrams, which are subsequently merged into state diagrams showing the complete lifecycle of an object. This approach is still feasible in UML and also available in the UML meta model [11].

Nevertheless, it seems a rather unnatural approach for formalizing use cases. Since a use case is a hierarchical collection of scenarios, these would have to be separated, formalized separately in interaction diagrams and then merged again into a single state diagram.

3 Use Case Documents

3.1 Examples

The structure of the following use case was inspired by Alistair Cockburn's use case template [2]. The example serves to clarify the points we make below.

| | |
|-----------------------|--|
| Name | Borrow Book |
| Goal | This use case describes how a library user selects and then borrows a book from the library. |
| Preconditions | None |
| Postconditions | The user is registered as the borrower of the book in the library system. |

TABLE 1. Borrow Book Use Case

| | |
|------------------------------|---|
| Main Success Scenario | <ol style="list-style-type: none"> 1. The user selects the search function from the main menu. 2. The system displays the search form. 3. The user enters the title of a book (possibly using wild-cards). 4. The library system presents a list of all matching books 5. The user selects a book. 6. The system displays the detail view for this book. 7. The user selects borrow from the menu for this book. 8. The user is already logged in. The system issues a message to the archive that the book is reserved for the user. |
| Extensions | <p>4a) There are no matches to the query.</p> <p>4a1) The system returns to the main screen.</p> <p>8a)The user is not logged in.</p> <p>8a1)The user logs in as described in Log in.</p> |
| Variations | <p>3a) The user enters the name of the author.</p> <p>3b)The user selects the author from an author list.</p> <p>3b1)The clicks on “select author”.</p> <p>3b2)The system displays a selection list of all authors.</p> <p>3b3)The user selects an author from the list.</p> |
| Included Use Cases | Log in |

TABLE 1. Borrow Book Use Case

| | |
|------------------------------|---|
| Name | Log in |
| Goal | This use case describes how a library user logs into the system to prove his identity. |
| Preconditions | None |
| Postconditions | The user is logged in |
| Main Success Scenario | <ol style="list-style-type: none"> 1. The user selects log in from the main menu. 2. The system asks the user for his login name. 3. The user enters his login name. 4. The system verifies login and password. They are ok. 5. The system logs the user on. |

TABLE 2. Log in use case.

| | |
|---------------------------|---|
| Extensions | 4a) The combination of login and password is not ok. 4a1) If the number of retries is not exceeded, repeat the use case from step 2. |
| Variations | None |
| Included Use Cases | None |

TABLE 2. Log in use case.

4 Use Cases to State Machines

Recent versions of UML [8] include a powerful state machine concept inspired by Harel's state charts [3, 4, 5]. Especially the abstraction mechanisms in the UML state machine formalism, e.g. nesting of states and stubs, allow us to map all the important elements of the use case template introduced in section 3 to state machines.

4.1 Main Success Scenario

First, let's consider the main success scenario of our use case. Each step in a use case corresponds to a message sent by an actor to the system or vice versa. The interval between two messages sent to the system is an abstract state of the system. As proposed by Rumbaugh [7], we denote all messages sent by the system as actions of the state. Each message sent by an actor is denoted as an event, causing a transition between two states of the system. The beginning of the use case is modelled by an initial state, the use case ends in a final state of the state machine. As intended in use case analysis, the main success scenario ends with the successful achievement of the goal [2]. Thus, the final state reached after the last step of the use case corresponds to successful completion. In section 4.3, we will model failures of the use case. The whole state diagram is encapsulated in a super state named after the use case for later reuse - to model relationships to other use cases. Figure 1 shows the state diagram corresponding to the *Borrow Book* use case.

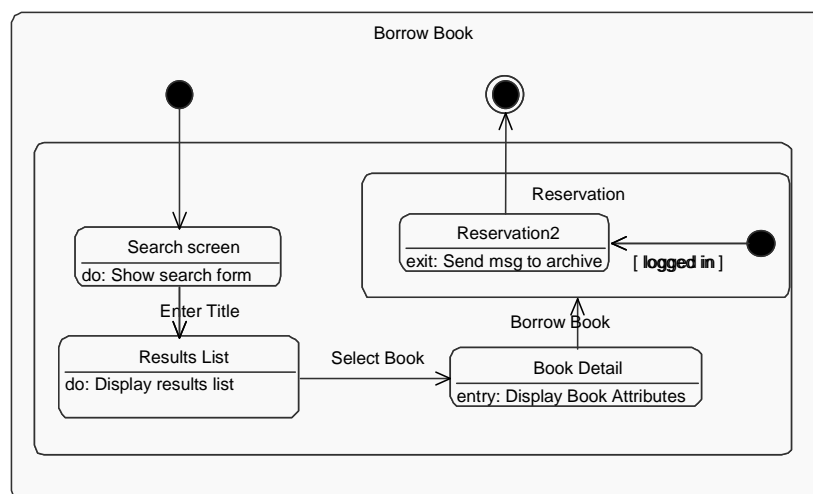


Figure 1: Basic state diagram capturing the main success scenario.

Each UML state diagram corresponds to an object. Since use cases describe the externally visible behaviour of the system as a whole, the state diagram corresponds to an abstract system object, e.g. the *Library*. The events shown in the state diagram, e.g. *Enter Title* and *Select Search*, are not necessarily methods of a concrete class - *Library* will probably be implemented by a large number of classes - but events the system understands. These map to statements in a test script for automatic system testing.

4.2 Variations

Variations in use cases are usually local alternatives for executing a step in the use case. In a state diagram these can be modelled as multiple paths connecting two states. In the simplest case, a variant can be represented as an additional link between the two states delimiting the corresponding step in the main success scenario. Intermediate states may be needed to model more complex variants.

In our example, *Enter Author Name* is an additional transition from *Search Screen* to *Results List* formalizing variant 3a). Variant 3b) needs more elaborate treatment, since an intermediate action of the system (displaying the *Authors List*) is required. This variant is modelled by the transitions *Select Authors List* and *Select Author* and the intermediate state *Authors List*. The modifications are shown in Figure 2.

4.3 Extensions

An extension usually describes a backup action for completing a subgoal in a use case, in case the default action for reaching that goal fails. A special case of this is when a subgoal fails, because a precondition does not hold as in the example's extension 8a). A modular way to handle extensions is to specify them using substates of the state representing the corresponding step in the main success scenario. Exten-

sion 4a) is an exception, where we abandon the goal of the use case. No book is found and the library system goes into an error state.

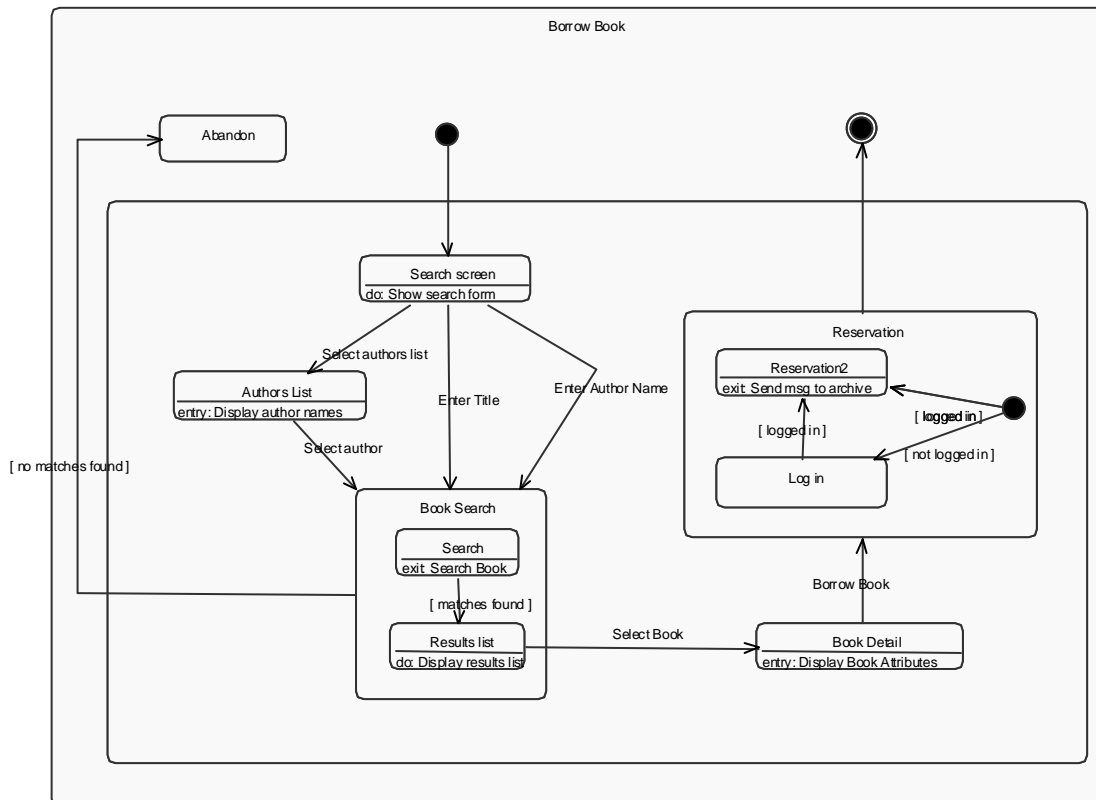


Figure 2: State diagram with variations and extensions. The *Log in* state is currently a placeholder. In section 4.5 we will describe how the invocation of the *Log in* process is modelled.

4.4 Preconditions and Postconditions

The Preconditions and Postconditions sections of the use case template allow to specify the contract of the use case [1]. Preconditions describe verifiable conditions, which must hold before the execution of the use case. In our use case template [2] postconditions are divided into *Success end condition* and *Failed end condition*. These sections define constraints which must hold upon successful or unsuccessful completion of the use case, respectively. We model the preconditions of the use case as constraints on the first state representing the use case. The upper part of Figure 3 shows how a precondition on a state can be modelled in UML using a superstate with two substates. We model postconditions of the use case as constraints on the final

state representing the use case. The lower part of Figure 3 shows how a precondition on a state can be modelled using a superstate with two substates.

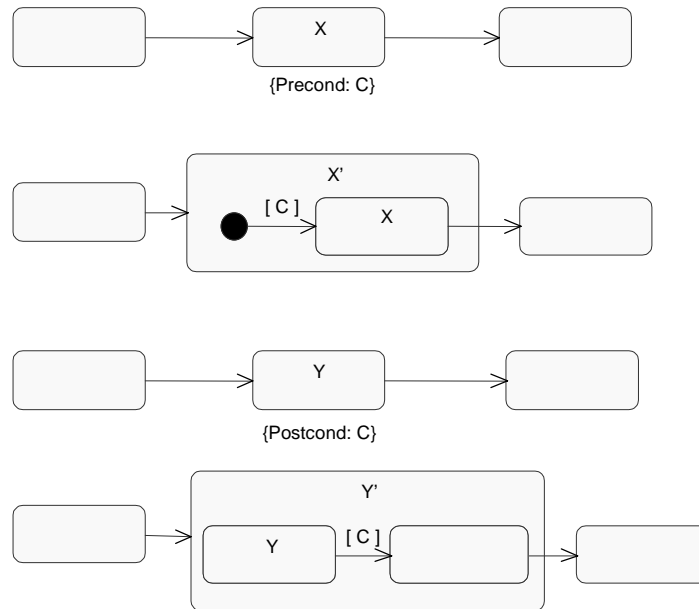


Figure 3: Modelling Pre- and Postconditions using UML State Diagrams

4.5 Subordinate Use Cases

UML defines a mechanism for including one use case as a subfunction in another use case. In use case diagrams this is expressed by a use case relationship of stereotype *include* [8]. The same information is present in the subordinate use case section of our template. In our example, the *Borrow Book* use case includes the functionality of the *Log In* use case. Figure 4 shows the state diagram for the *Log In* use case. The *Log In* use case has a successful final state where the user is logged in, and an unsuccessful end state, where the user has exceeded the maximum number of attempts to enter the valid password. To integrate the *Log In* use case with the *Borrow Book* use case we use two techniques from the UML state diagram notation:

- A submachine reference state allows to copy the state machine formalizing the subordinate use case (*Log in*) into the enclosing use case (*Borrow Book*).
- Stub states allow to connect the states in the submachine (*Log In*) to the right states in the enclosing machine (*Borrow Book*).

Figure 5 shows how the use case *Log In* is embedded into *Borrow Book*. If the user is not logged in at the beginning of the reservation state, the transition to the *Login Prompt* state is activated. This transition to an internal state of *Log In* is shown with the stub notation. The default (successful) exit of the login substate is connected to *Reservation2* and achieves the logged in condition needed by that state. Now that the logged in condition is assured, we can remove the label [logged in] from the transition.

The unsuccessful exit of login, formalized by the *Login Failed* stub is connected to the error state of the *Borrow Book* use case, which is called *Abandon*.

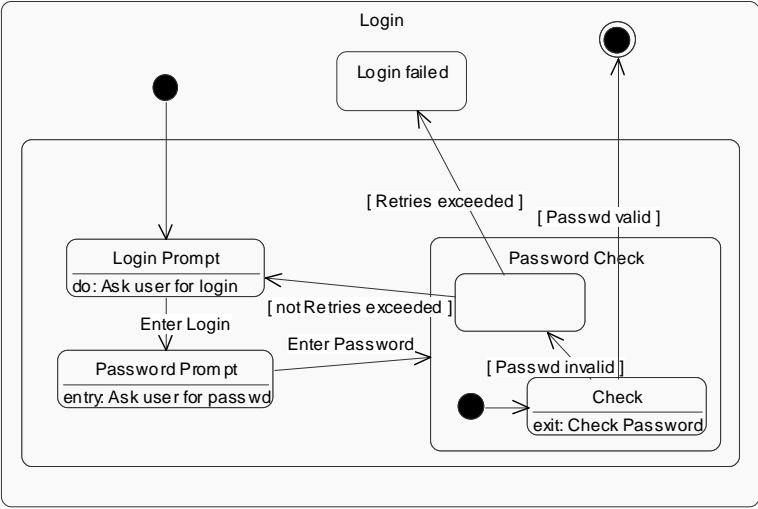


Figure 4: State machine corresponding to the log in use case.

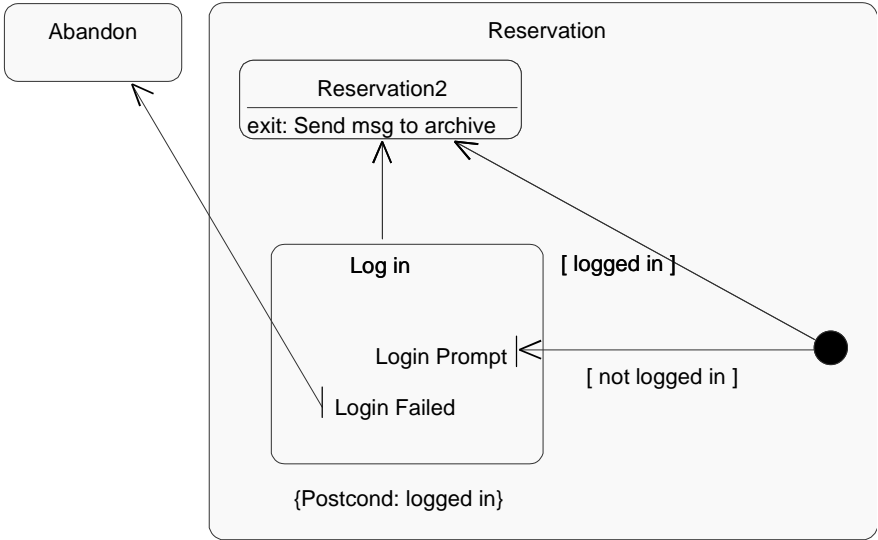


Figure 5: State machine extension showing the included use case login.

5 Integration of Use Cases

In scenario-based approaches to dynamic modelling it is a difficult task to integrate all the scenarios correctly into a state machine, because those states from the different scenarios must be identified which can be merged into a single one in the overall state diagram. Desharnais et al. have recently described a fully formal approach for scenario integration [10], which is however only applicable, if the states are fully characterized by formulae, a rather unrealistic assumption in applied software engineering.

We have shown in section 4 how a state machine can be constructed directly from a use case description. This leads us immediately to a consistent state machine for all scenarios covered by the use case. Thus, the direct transition from use cases to state machines has eliminated a large fraction of the necessary integration effort.

6 Using State Machines for Test Automation

An important step within thorough testing of object-oriented applications is the transition from testing individual methods to classes and subsystems. When focusing only on the individual method's input and output parameters you start missing important aspects of the overall behaviour of a class. State machines often describe the overall dynamic behaviour of a class or a system.

Schneider and Winters [13] describe how use case modelling can be applied on different levels of detail in system design, i.e. both the description of functional requirements from the user's perspective and the specification of a class or subsystem's API (application programming interface). Since we show how to derive state machines from use cases in general, the resulting state machines can be used as test models in several contexts:

- In *functional requirements testing* events in the state chart usually correspond to concrete user actions, e.g. "Enter Password". The test cases derived from the state chart can then easily be used to produce written test descriptions or to trigger the recording of GUI tests with a GUI test automation tool.
- In *unit or subsystem testing* events map directly on to function or method calls in most cases. If the syntax for events, actions and their signatures is specified accordingly, executable test cases could be derived automatically or semi-automatically from state machines.

The creation of test cases from state machines in its basic form is straightforward: Events and actions map on user actions or method calls, respectively. Siegel [15, pp. 195] describes a few rules for checking state-transition models before using them for testing. After applying these rules the state machine can be used to produce a sequence of test cases with a given coverage criteria, e.g. "every transition in the chart must be covered by at least one test".

If we apply the above coverage criteria on the state machine from figure 2 (ignoring the substates), we can derive the following three test cases:

1. Test case 1:

- Select Authors List
- Select Author
- Select Book
- Borrow Book

2. Test case 2:

- Enter Title
- Select Book
- Borrow Book

3. Test case 3:

- Enter Author Name
- Select Book
- Borrow Book

In this example it can be seen that the test cases so far do not differentiate between obvious and important cases: Entering search strings for titles or authors can result either in a *matches found* or *no matches found* state. This leads to what we have seen in section 4: semantic conditions must be included in the state machine which are meaningful to the user (e.g. *logged in* or *matches found*). The state machine model thus contains transitions labelled by these conditions. While we can have the system check the condition and thereby record coverage of a given test series, it is not trivial to enforce these conditions to create a set of test scenarios leading to a given coverage level. However, if we model explicitly, that a state achieves a certain condition, we can plan sequences which guarantee to traverse a certain transition with that condition. This is a topic we are currently investigating.

7 Conclusion

In our paper we have discussed various approaches to transform use cases into formal representations with strict semantics. Unlike interaction and activity diagrams state machines - represented as state charts - have the necessary qualities to be easily used for dynamic software design and test automation. Our approach to map use cases onto state charts shows the following properties:

- It integrates scenarios of a use case into one state chart.
- It allows to map all relevant elements of a use case into the chart.

State charts are a powerful means for modelling dynamic behaviour and deriving test cases; use cases are an essential tool in today's requirements analysis. Enabling the transformation between the two can be another step towards a repeatable and traceable software development process.

8 References

- [1] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997.
- [2] Alistair Cockburn. *Structuring Use Cases with Goals*. Journal of Object-Oriented Programming, September and November 1997 issues.
- [3] D. Harel. *Statecharts: A Visual Formalism for Complex Systems*. Science of Computer Programming, Vol. 8, 1987.
- [4] D. Harel. *On Visual Formalisms*. Communications of the ACM, Vol. 31, No. 5, Pages 514-531, May 1988.
- [5] D. Harel, Michal Politi. *Modeling Reactive Systems With Statecharts: The STATEMATE Approach*. McGraw-Hill, New York, N.Y., 1998.
- [6] Ivar Jacobson, Magnus Christerson, Patrick Jonsson, Gunnar Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Wokingham, England, 1992.
- [7] James Rumbaugh, Michel Blaha, William Premerlani, Frederick Eddy, William Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, N.J., 1991.

- [8] James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Mass., 1999.
- [9] Martin Fowler. *Use and Abuse Cases*. Distributed Computing, April 1998.
- [10] Jules Desharnais, Marc Frappier, Ridha Khèdri, and Ali Mili. *Integration of Sequential Scenarios*. IEEE Transaction on Software Engineering, Vol. 24, No. 9, September 1998.
- [11] Object Management Group. *Unified Modeling Language Specification*. Framingham, Mass., 1998.
- [12] Russell R. Hurlbut. *The Three R's of Use Case Formalisms: Realization, Refinement, and Reification*. Technical Report XPT-TR-97-06, Expertech, Ltd, 1997.
- [13] Geri Schneider, Jason P. Winters. *Applying Use Cases: A Practical Guide*. Addison-Wesley, 1998.
- [14] Brian Marick. *The Craft of Software Testing*. Prentice Hall, 1995.
- [15] Shel Siegel, Robert J. Muller. *Object-Oriented Software Testing: A Hierarchical Approach*. John Wiley and Sons, 1996

Modelling Dynamic Behaviour Based on Use Cases

Peter Fröhlich and Johannes Link

ABB Corporate Research
Speyerer Straße 4
69115 Heidelberg, Germany

Email: peter.froehlich@de.abb.com,
johannes.link@andrena.de

Motivation

- Use Cases
 - Hierarchical collection of scenarios describing functional requirements
 - Commonly used for Requirements Capture
 - Vision: Development process driven by use cases
 - From Use Cases to Design
 - Testing based on Use Cases
 - ⇒ Controlled process requires formal mapping to representation with stricter semantics
- Possible Target Formalisms
 - Interaction Diagrams
 - Activity Diagrams
 - State Machines
- State machines seem the most suitable approach

Previous Work

- Use Cases to Activity Diagrams
 - Activity Diagrams
 - Recent addition to UML
 - Good means for communication with customer
 - Disadvantages of Activity Diagrams
 - Missing connection to OO-Paradigm
 - Models central control structure instead of message-based interaction
 - No distinction between normal and abnormal behaviour
- Use Cases to Interaction Diagrams
 - Interaction Diagrams
 - Already in OMT
 - Disadvantages of using Interaction Diagrams
 - One Interaction Diagram is one scenario
 - ⇒ Scenarios which are related in the use case are separately modelled in different diagrams
 - ⇒ Must be merged formally later: “Scenario integration” is necessary

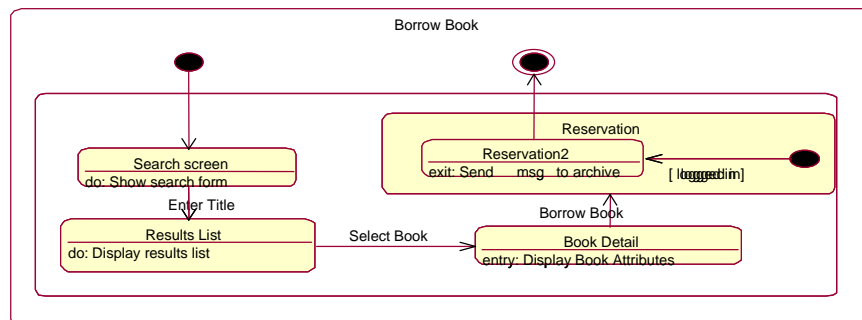
Use Cases to State Machines

- Advantages
 - Expressive, hierarchical formalism
 - Integration of all scenarios of a use case in one state chart. Relationships among scenarios are defined by the formalism.
 - Nested states allow to model *include*-Relation among use cases explicitly
 - Conditional transitions allow to model Pre- and Post-conditions
 - Events in state charts map to methods in class definitions
- Application Areas
 - Design of dynamic object behaviour
 - Easy derivation of test cases

Example

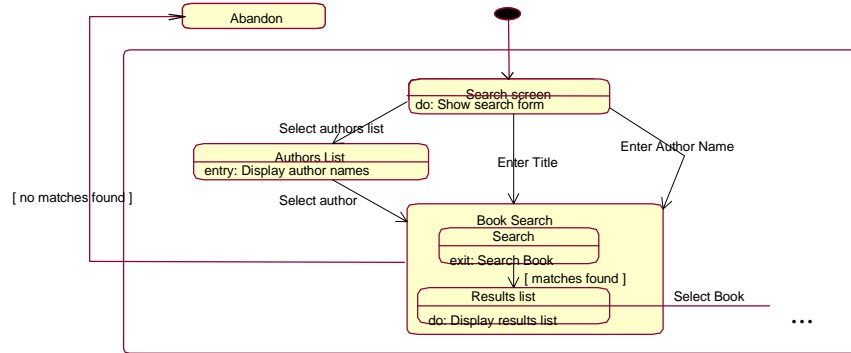
- Borrow Book
 - Main Success Scenario
 1. The user selects “ search” from the main menu.
 2. The system displays the search form.
 3. The user enters the title of a book.
 4. The system presents a list of all matching books.
 5. The user selects a book.
 6. The system displays a detail view for this book.
 7. The user selects borrow from the menu.
 8. (User logged in). The system reserves the book.
 - Extensions
 - 4a) There are no matches to the query.
 - 4a1) The system returns to the main screen.
 - 8a) The user has to “Log in” to borrow the book.
 - Variations
 - 3a) The user enters the name of the author.
 - 3b) The user selects the author from an author list
 - Included use case: Log in
- Log in
 - Pre-condition: not Logged in
 - Post-condition: Logged in

Mapping to State Machines (1) Main Success Scenario



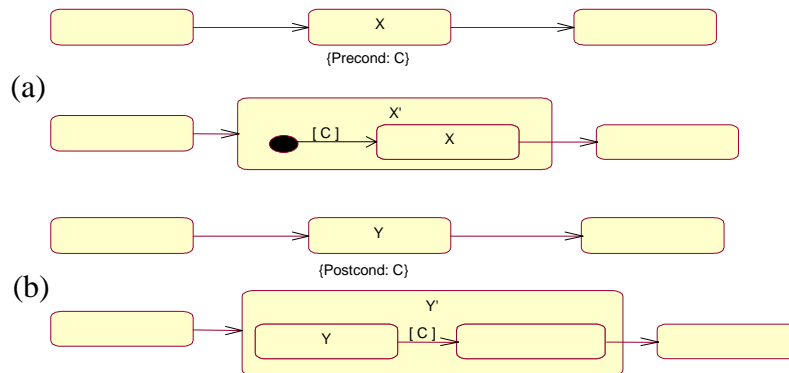
- Use case -> State (with further internal states)
- Use case start -> Initial state
- User step -> Event (e.g. “Select Book”)
- System step -> Action of a state (e.g. “Show search form”)
- End of use case -> Final state
- Step with precondition -> Substate with conditional transition (e.g. “logged in”)

Mapping to State Machines (2) Variations & Extensions



- Variations -> Alternative transitions
 - Simple Transition (e.g. "Enter Author Name")
 - Complex Transition with intermediate states (e.g. Author List)
- Extensions -> Substates with conditional transitions depending on success or failure
 - Additional error states (e.g. "Abandon")

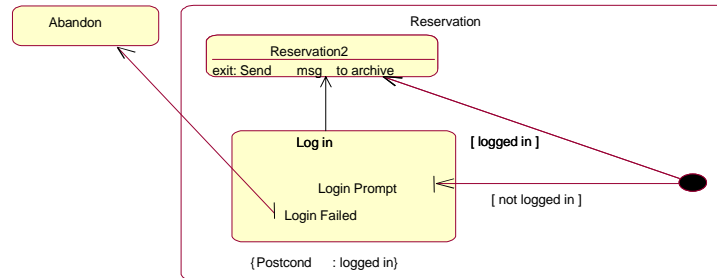
Mapping to State Machines (3) Pre- and Post-conditions



- Pre-condition of a use case -> Pre-condition of the first state
- Split state with pre-condition as in (a)
- Post-condition of a use case -> Post-condition of the last state
- Split state with post-condition as in (b)

Mapping to State Machines (4) Subordinate Use Cases

- Assume, state machine for Log in exists



- Include statement -> Submachine reference state
- Stub states are used to connect the enclosing state machine (for Borrow Book) to the included state machine (for Log in)

Test Automation

- State machine describes the overall dynamic behaviour
 - State machines as test models for Functional Requirements Testing and Subsystem Testing
 - Events: user actions or method calls
 - Algorithms exist to systematically derive tests with given coverage level
 - Possible Extension based on Pre- and Post-conditions: Make sure the test sequence satisfies preconditions of all involved states

Conclusions

- Use Cases can be formally mapped to state machines
 - The mapping allows for the integration of all elements of a typical use case document
 - In contrast to previous approaches, the relationships among all scenarios of a use case are represented formally
- The resulting state machines can be used in design and testing

A post-mortem analysis of a semi-successful client server system test project.

Mats Grindal, Enea Test
Enea Data AB
www.enea.se
Box 232, S-183 23 Täby, Sweden
Tel: +46-8-50 714 000
Fax: +46-8-50 714 040
E-mail: magr@enea.se

©1999 Enea Data AB
All rights reserved

Abstract

Using the results and experiences from a client server system test project, this paper tries to answer the questions "What worked well?", "What didn't work?" and "What can be done to improve the next project?".

The planned duration of the system test project was 10 months, from the first planning to the delivery of the last customer system. The contents of the project included a Y2K test, a number of new features for old customers and the customization of the system for one new customer. Each one of the customers should receive a unique configuration but some of them differ in only minor parts, leaving a total of four clearly different configurations that were to be tested.

A typical customer site includes a server or a group of clustered servers. Via some kind of network up to a few thousand clients are connected to the server. The clients are distributed geographically.

The major issues for this system is security, reliability and load handling. Although speed is also important, the manual part of the transaction is more than 5 times longer than the system part.

The key lessons learned from this study are that integration test organisation and the use of a test co-ordinator to help optimise test effort will help system test greatly. The hand over of deliveries from integration test to system test also is of great importance to the end result.

Contents:

| | | |
|-----------|---|-----------|
| 1. | Disposition | 4 |
| 2. | Scope and background | 4 |
| 3. | Organisation and procedural issues | 5 |
| 3.1 | Integration test..... | 5 |
| 3.2 | Test co-ordinator | 6 |
| 3.3 | Delivery Projects | 7 |
| 3.4 | Resource allocation | 7 |
| 3.5 | The role of system test..... | 7 |
| 4. | Planning phase | 8 |
| 4.1 | System test time budget..... | 8 |
| 5. | Preparation phase | 9 |
| 5.1 | Non-frozen requirements..... | 10 |
| 5.2 | Quality of test cases..... | 11 |
| 5.3 | Inspection | 12 |
| 6. | Execution phase | 12 |
| 6.1 | Hand-over meeting | 12 |
| 6.2 | Regression strategy..... | 13 |
| 6.3 | Non-functional testing..... | 14 |
| 7. | Reporting phase | 15 |
| 7.1 | Error reporting process..... | 15 |
| 7.2 | Basis for delivery recommendation..... | 16 |
| 8. | Conclusions | 16 |
| 9. | References | 17 |

1. Disposition

The disposition of the paper is to first give some background information on the project itself in order to put this study in perspective. Then follow five areas of study, the first related to the organisation of the project and the remaining four each related to each one of the four phases of the system test project i.e. planning, preparation, execution and reporting. For each area of study relevant results of the project are discussed and where appropriate improvements are outlined. The last section of this paper summarises the most important improvements suggested throughout the paper.

2. Scope and background

The studied system is a client server system. A typical customer site includes a server or a group of clustered servers. Via some kind of network up to a few thousand clients are connected to the server.

For server HW, off-the-shelf mainframe computers are used. Operating system and database applications are bought externally and only the application SW (10 million lines of code) is developed in house. Both client HW and SW (1.2 million lines of code) are developed in house. But the use of standard HW products facilitates buying external operating systems.

The goal of the development project of which the studied system test project belongs to was to build a new release of an already existing client server system. Functions and features in this new release included:

- Securing the millennium shift for the system.
- Developing a new custom-designed version of the system for one completely new customer.
- Totally re-designing one old sub-system.
- Adding of a few pieces of old functionality to new customers.
- Making a number of small improvements across the whole system
- Performing error corrections on errors in previous releases.
- Improving the boot code of the client.
- Building a HW prototype of a new client.

The whole system was affected by these changes so from a system testing perspective there were no obvious shortcuts. However, the transaction engine and large parts of the other core functionality in the server came out almost untouched.

Apart from functionality the major issues for this system are security, reliability and load handling. Although speed is also important it is not so critical because the manual part of the transaction takes more than 5 times longer than the system part.

Each one of the customers should receive a unique configuration but some of them contain only minor differences, leaving a total of four clearly distinct configurations. Although each customer configuration should be tested, the emphasis of the testing could be concentrated on these four different configurations.

In the original time plan there were 10 months allocated for the system test.

3. Organisation and procedural issues

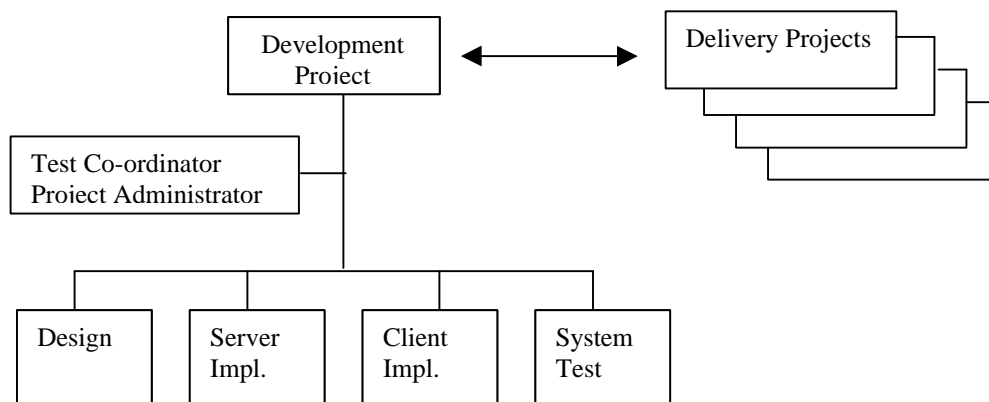


Fig. The overall structure of the project

As can be seen in the picture above, the system test sub-project was one of four sub-projects within the development project. Customers to the development project were four delivery projects, which in turn served the customers.

3.1 Integration test

The only organisation dedicated solemnly to testing was the system test sub-project. Module testing was performed within the implementation projects with the sub-project managers responsible. For the integration testing the test co-ordinator was responsible since the activities involved resources from both implementation sub-projects. The idea was that the test co-ordinator in co-

operation with the sub-project managers of the implementation sub-projects should build integration test groups, which should be led by the test co-ordinator. This scheme had two major drawbacks. Resources for integration testing were not always available due to prioritising of implementation activities, and those that were available were not always motivated to perform testing. The result was that the integration tests were not always well performed, which in turn affected system test progress.

One way to solve this problem would be to create a separate integration test sub-project with its own resources. People working with integration test will be better motivated since they will have integration as their primary task. With this solution, testing skills can also be permanently associated with integration tests. This solution has yet another advantage, as will be explained in chapter 6.1 the role of system test can be better focused.

3.2 Test co-ordinator

On the positive side, the existence of the test co-ordinator gave the system test sub-project a single speaking partner for plans and results of tests of all levels of the whole project. For instance it was possible to identify areas where the amount of system testing could be adjusted, depending on the results from integration tests. As been mentioned in the previous section, integration tests were not always performed according to plan adding more tests to be performed in system test.

To some extent the test co-ordinator also offered the opportunity to exchange test cases beforehand between the integration and system test phases in a controlled way. But as integration test resources were not always secured, the exchange of test cases almost always was to the disadvantage of system test, with planned integration tests being moved to system test.

It is my belief that it is an important, although difficult role of the test co-ordinator to find ways of cutting test time by exchanging test cases over the traditional testing phases. For instance some activities traditionally referred to a system test activities could very well be performed during integration tests if all required system components already are in place. Correspondingly some module tests could very well wait until system test if the cost of implementing the extra testware required i.e. stubs and drivers, exceeds to the cost of waiting until the real functions are implemented. The idea behind these thoughts is presented in [1].

3.3 Delivery Projects

Another positive aspect of the organisation used was the use of delivery projects. Each customer had a delivery project acting as interface between the customer and the development project. This approach gave the system test sub-projects as well as the other sub-projects an opportunity to work without being constantly interrupted by customers asking for help, expressing opinions and so on. Another positive thing was that the same persons handled the whole information flow between the development project and the customer. Conflicting decisions could thus be discovered early and kept to a minimum.

3.4 Resource allocation

For the system test sub-project the resource allocation worked well. A project-planning tool was used to communicate the resource needs to the line organisation. There were several different projects competing for the same line resources. Each week the needs of all the various projects were added up in the project planning tool and the result was returned both to the line organisation and the different project managers.

The total resource need more than five weeks ahead was used by the line manager to plan training, vacation, new employments etc. No explicit granting of resources were made on these “long term needs”. For “short term needs” i.e. resource needs within a five week time frame, these needs were discussed and negotiated between the line and the project managers at a weekly meeting right after the result from the resource need summation was available. The choice of a five-week time window, is a balance between looking as far ahead as possible and the correctness of each project's needs. Several of the parallel test projects competing for the same resources have had frequent and major re-plannings. The benefit of using longer time frames is therefore questionable. A successful combination of managers (both project and line) demonstrated the ability to see the overall picture, thus in consensus dividing resources among the projects.

3.5 The role of system test

The last thing to be commented on regarding the organisation, is the fact that system test was part of the development project. This led to a severe conflict of interest. As part of the development project system test automatically became partly responsible for the product quality. On the other hand system test was also expected to act as an independent quality controlling body. During of the

two first customer deliveries system test made the mistake of going along with the rest of the project adopting a “constructive” thinking. After some weeks of system test the number of open error reports had been reduced which was interpreted as a sign of good system quality. The system quality measured in open error reports together with the assumed quality responsibility gave the testers no incentive to look for problems beyond the obvious. This led to a recommendation of delivery for these two customer systems, but when the systems reached the customers the false conclusion was soon evident. The customers soon found several grave problems that had been subsequently missed by testers. System test learned from its mistake, not recommending further customer deliveries. Which in turn caused the undesired situation that systems were eventually delivered to the customers without the recommendation from system test.

To solve this conflict of interest, the same solution as outlined above, integration test being a separate sub-project, can be used. The advantage would be that the integration sub-project could take the product quality responsibility, maintaining a constructive approach and system test can focus mostly on the auditing function, where a more destructive approach is beneficial. For further comments see also chapter 7.2 where this issue is further commented on.

4. Planning phase

The most important factor in the planning phase of the whole project affecting the result of the project, is the method of identifying the work to be performed and estimating the time needed to do this.

4.1 System test time budget

The method used by this company for creating a time budget for a project, starts with customers ordering new features and other work to be done. These orders are refined into *work orders*. A work order can also originate within the company. The redesign of one of the sub-systems and the correction of error reports from previous releases are examples of internal work orders. System specialists then analyse each work order to assign a time budget for all the necessary work. This time budget includes time for project management, design, implementation and test. When there is enough total time worth of pending work orders or for any other reason a good collection of work orders awaiting implementation, a new development project is formed.

The rule of thumb to determine the amount of time available for system testing is that 15% of the total time for a work order should be used for system testing. In addition to these 15%, extra time should be allocated for regression testing of old functionality. The amount of time budgeted for regression test is a function primarily of how many customers that are included in the project. For the system test sub-project covered in this report this rule of thumb was violated resulting in only 10% of the total time for the work orders being available for system test. If the extra time for regression testing is included the total time available for system testing was 14 %. The reason behind this violation was primarily lack of understanding of testing issues from the system specialists' part. The consequence of this is naturally that testers need to be involved earlier in the time budget process and/or testing knowledge of the system specialists' needs to be improved.

Besides participating early in the time budget process, improvements can also be made on the rule of thumb itself. It is obvious that a software change that is isolated requires less testing effort than an equally big change that is distributed over the whole system. Trying to find suitable criteria to classify the degree of isolation and thus indicating testing effort have to be performed.

The actual result i.e. the ratio of system test time to total time used was 10.6%. The main reason for this being such a low figure was that the implementation projects grew but due to lack of available resources on the system test side the corresponding growth of system test could not be maintained.

It is however hard to draw any valid conclusions from the low number. It is evident that system test would have benefited from more time but it is impossible to say how much more time that would have been needed. The main reasons are that much of the system test time was spent on integration testing and that the system tests in general were inefficient.

5. Preparation phase

The preparation phase of the system test project was mainly aimed at writing test cases, test instructions and preparing test data. Out of the 10 calendar months planned project duration, 4 calendar months were planned for test preparation. The planning phase was completed on time with only minor deviation from the original budget.

As far as the quality of the work is concerned it was maybe at little worse than expected but that can be explained by the large number of inexperienced testers involved, which suggests that the ratio in calendar time of $4/10 = 40\%$ is

roughly OK. Noteworthy is also that there was roughly the same amount of people involved throughout the whole project, even though more overtime had to be used in the execution phase. By lowering this ratio somewhat this conclusion should be fairly right even if “man-time” is used.

The three key areas related to the preparation phase that will be discussed in this paper are requirements and change request handling, test case quality and reviews. There are of course many more aspects that could be dealt with but these three areas are judged to be the biggest contributors to the end result.

5.1 Non-frozen requirements

The largest problem encountered during the preparation phase was non-frozen input documents. Even at the end of the design project there were still a number of requirement specifications, which had not been approved. The immediate consequence of this kind of problem is that constructed test cases will be incomplete and/or incorrect. During the course of test execution a large number of error reports have been written only to end up in a “No Action” state due to this. Unfortunately it has also happened a few times that tests have been missed or passed on incorrect grounds. These problems were then found and reported by customers.

It is inevitable that requirements and the corresponding test cases need to be modified after the preparation phase is finished, so we need to develop a way of handling such cases. A way to improve matters is as a first step to schedule the finish (release) dates of new/modified requirement specifications. This document release plan is then made public and used for planning dependant activities in later sub-projects. All changes to the plan must be announced as soon as they are known so dependent activities can be rescheduled. Step number two is that the release of each requirement specification must be communicated and used as signal “go ahead”. The third part of the improvement includes some tuning (and clarification) of parts of the change request routine.¹

- Each change request must clearly indicate which requirements are concerned.
- No change to an already released document is allowed unless there is an approved change request describing the change.
- Change requests need to be approved both by the project and by a representative of the concerned customer(s).

¹ At this company there exists a seemingly working change request routine, but approved changes takes very long time to be entered into the requirements specifications, and the traceability is not working completely

- As soon as a change request has been approved, the requirements specification should be updated to a new preliminary version with a note in the revision history indicating that there is an approved change request overriding parts of the requirements, unless the change itself is entered in the document

5.2 Quality of test cases

For several testers this project was their first meeting both with testing and with this particular system. As there was not enough time to teach both testing skills and system knowledge, the latter was focussed on under the assumption that it is better to have testers that can work the system than having an optimised test suite that no one can run. The end result was that around 1600 test cases were written which roughly had a one-to-one correspondence with the new and changed requirements in this project. From a testing perspective this test suite had several weaknesses:

- Simple techniques like boundary value analysis were not used, leading to “inefficient” test cases.
- As requirements were mostly about functionality, several important system characteristics such as security, load handling, stress, reliability etc. were totally or partly missed during preparation.
- No regression tests were prepared.

When these problems surfaced during test execution phase, some measures were taken to compensate for these weaknesses. Boundary value analyses was discussed with the testers and the testers were asked to change test cases on the fly. This was a feasible solution as each test case contained the original requirement text. In the absence of clear performance requirements, some rudimentary benchmark tests were developed, and a checklist for a regression test was produced. The checklist contained no specific test cases instead it listed some base functions that were to be checked for each system.

As a result of the lack of testing skills a whole education package has been planned and will be held this autumn. First a regular test course will be held and then a number of workshops will be run in which company specific problems will be addressed. The result from each workshop will form the framework of improvements that is needed in the system test organisation.

5.3 Inspection

The last key area to be discussed within the preparation phase is inspection. Although inspections were expected to be performed, the inspection activity was never visualised explicitly in the time plan. As a result a tough inspection race had to be performed during the last two weeks of preparation, which meant that at the end of the two-week period reviewers were less motivated to do a good job. The inspections also consumed more time than what was left, leading to the need for overtime work as described earlier. As testers were quite unfamiliar with the system external help during the inspections would have been valuable. By the time the inspections were started the pressure on the design and implementation sub-projects was very high resulting in great trouble finding such external help.

Improvements on these issues start with including testers in the requirement process; this both strengthens the requirements process itself and educates the testers. Review activities need to be started earlier to even out the workload and increase the chance of participation of developers in the inspection process. Finally improvements should also be done in the inspection process itself; for instance more elaborate checklists and the start of data collection for future use.

6. Execution phase

The execution phase of the system test sub-project was originally planned to last 6 calendar months but due to a number of factors the actual time spent in the execution phase was increased to 9 calendar months. The most important reason for this was the low quality of the system when it was delivered to system test. Since large parts of the integration test had been missed, many more errors than anticipated remained in the code, hampering system test progress. Not only there was an extra overhead in finding and reporting errors but there were also a large number of extra deliveries performed. These extra deliveries put focus on the lack of regression strategy.

6.1 Hand-over meeting

The most important lesson from this is that system test needs to control what is delivered to it. Hand-over meetings were held once for each customer system at the first delivery of that system from integration to system test, but the information presented at these meetings was far from complete. There were no specified quality criteria to meet, so when system test tried to execute the right

to reject the delivery, there were big discussions which ended in system test not formally accepting the delivery but starting testing anyway.

The main purpose of the delivery meeting should be to judge the quality of the delivery thus being able to decide whether or not it is worthwhile to start the next phase. To increase the control two simple measures are to improve the written information passed from integration test to system test and to define acceptance criteria better. Valuable information about the deliverable include:

- What was planned to be implemented and what really was
- What was planned to be tested and what really was
- Outstanding errors
- Other information

It is my belief that a strong delivery acceptance process cannot rely only on information from the supplier. Instead the receiver needs to be more active. Therefore a trial period of 1-2 days is suggested. Thus the acceptance from the meeting is changed to an acceptance to start a system test trial period. The same fixed set of test cases is then used at every delivery, adding objectivity to the acceptance decision. Not until all test cases in the trial period have passed has the delivery been formally accepted and the real system test can start. A spin-off effect is that this set of test cases is also the first candidate for automation.

6.2 Regression strategy

The large amount of deliveries that has passed through system test is too large for almost any testing organisation to handle with good quality of testing. The number of deliveries needs to be significantly reduced but even then good regression test strategy is crucial to success. In this project there were no automated tests and no real definition of regression tests. To cope with the situation, a checklist was developed including only the most basic functionality that could be tested manually by one person during one day. This checklist was used on all deliveries going out to customers. It caught a few errors but given the time to prepare, a much better strategy inspired by [2] would be to create three sets of tests.

The first set of tests should include testing of all basic functionality that should be checked every time. Examples of such functionality are loading of new applications, exercising basic functions, running common transaction types, production of standard lists etc. All test cases in this set should be run for each new delivery. If it is possible, this set of tests should be automated or at least

semi-automated. This set of tests can be the same set of tests as described in the trial run in section 6.1.

The second set of tests is a large set of test cases from which different test cases are chosen different times. Before the regression test is started a reasonable amount of test cases *across the whole system* should be identified and prepared. If automation is considered the total amount of test cases within this set should be divided into subsets where each subset is automated independently. One or several subsets are then identified to be run, during each regression test. The key idea is that after the whole project all test cases in the whole original set have been executed at least once.

The third set of tests changes dynamically over time and focuses on areas where large changes/corrections have been made recently. Relevant test cases are taken from the test case specification and executed manually. Since the contents of this set changes over time it is not a good investment to automate them unless this task is easy.

Another important lesson from this is that the quality of integration testing needs to be improved, which again suggests integration tests being run in a separate sub-project.

6.3 Non-functional testing

As have been mentioned earlier, the focus of system test was on functionality. Issues like reliability, load handling, stress, performance and security were not dealt with to a necessary extent from a testing perspective. The main reasons for this incorrect focus were lack of requirements on these issues, inexperienced testers and the combination of initial bad quality and lack of time.

Some minor benchmarking activities have been performed. The results will be used in later projects to compensate for the lack of requirements. An education program has been launched for the testers and there are activities in other parts of the company to support the quality question. So in terms of learning from mistakes all these things will be improved in the future.

As far as the final result, we were in the fortunate situation that system core remained virtually untouched by this development projects so issues like reliability, load handling, stress, performance, security etc were areas that were

working well from the beginning. This claim is also supported by the fact that no major problems within these fields have been found by the customers so far.

7. Reporting phase

There are three main reporting tasks to be carried out by the system test sub-project.

- Supply the project with continuous information on the quality of the test object. This information is used by the project management to make tactical development decisions. The key sources of information are the error reporting tool and the weekly status report from the system test sub-project.
- Supply the delivery projects with information on the quality of a particular customer system at the time of delivery. This information is used by the delivery project to make a decision about actual customer system delivery. Key sources of information are the error reporting system and test reports.
- Supply the line organisation and the project management with information on the results of the project not only in terms of product quality but also economy, efficiency etc. This data is used in the continuous improvement program.

All these three functions worked well but there are two issues connected to the reporting functions that require more attention.

7.1 Error reporting process

The error reporting process at this company had been built around a small in-house developed tool. This tool was never intended for this type of large projects so the consequence was that "work-arounds" have been invented locally to get around the shortcomings of the tool. At the moment there are six different documents existing at the company describing parts of the error reporting process and the use of the tool. These documents lack some vital information and even contain some minor contradictions, leading to misunderstandings. In the studied project the problems with the error reporting tool and the associated process was soon evident so a separate log was kept by the system test sub-project to ensure that no error report would be lost. The lesson learned here is that the company has outgrown the small tool that once was enough for its needs. When a new tool has been acquired the error reporting process also needs to be adjusted to the needs of large projects as well as the abilities of the acquired tool

7.2 Basis for delivery recommendation

In the test report used at the delivery meetings the system test project should give a recommendation whether or not to go ahead and deliver the software to the customer. This recommendation should be the view of system test and reflect the system quality experienced during testing. As system test prior to the delivery meeting had been working in close co-operation with the implementation projects trying to make things work, the mental effort of suddenly being the independent quality judge became too large. In retrospect the recommendations of the two first customer deliveries were not accurate.

This problem has also been discussed in chapter 3.5 and in that chapter some organisational changes were suggested. However there is also a need for more objective metrics. At the moment the only measured variable is the number of outstanding error reports. Coverage metrics also needs to be used in order to relate the number of errors to the test effort. Metrics should also be used to indicate the total impact of the outstanding errors for the customer. The company has acknowledged the need for metrics but the actual choice of metrics is under investigation.

8. Conclusions

A quick evaluation of the system test results will probably not show so much on the positive side, but if root causes are examined more closely, a different picture emerges. It is true that there is a large room for improvement in the within the system test department and in future system test projects, but many of the causes for the delays are beyond the control of system test. If taken into account that most of the system test resources were very inexperienced in the beginning of the test project, the end result actually looks quite good. Of course such a claim should not be made outside the testing department in an attempt to cover the situation, but instead it should be used inside the testing department to show that the testers actually have done more than what could be expected of them with their starting points.

As many of the issues affecting the result of system test lie outside the control of the testing project, the lessons learned will also include several things outside the normal scope of system testing. This paper is concluded with a list of the most important lessons learned from the system test perspective.

- Delivery projects are good, as they act as buffers between customers and development/test projects.
- The strategic role of the test co-ordinator should be emphasised to make most out of the possible co-operation between the different levels of testing.
- Integration test should be a separate sub-project to visualise this activity and improve the resource allocation.
- 30-40 % of the total test time should be planned for preparations.
- Inspections need to be started earlier and should be visualised in the test plan.
- The delivery process to system test should allow for a trial period before system test is started, and should grant system test the authority to reject a delivery.
- A thorough regression strategy is needed which deals both with basic functionality and recently changed areas of the code.
- More test focus is needed on stress, performance, load handling, security etc.
- A new tool for error reporting needs to be acquired. It should be designed for more complex projects and easily adaptable to changed project requirements.

9. References

- [1] Marick Brian – New Models for Test Development.
Conference Proceedings Software Quality Week 1999.
- [2] Kaner, Falk, Nguyen – Testing Computer Software, 2nd edition
ISBN 0-442-01361-2

A Post-Mortem Analysis of a Semi-Successful Client Server System

Mats Grindal, Enea Test
Enea Data AB
www.enea.se
email: magr@enea.se

© Enea Data AB A post-mortem analysis... 1.0 990919 Mats Grindal 1 (11)

ENE A DATA

Outline

- Project Scope
- Organisational issues
- Planning Phase
- Preparation Phase
- Test Execution Phase
- Reporting Phase

© Enea Data AB A post-mortem analysis... 1.0 990919 Mats Grindal 2 (11)

ENE A DATA

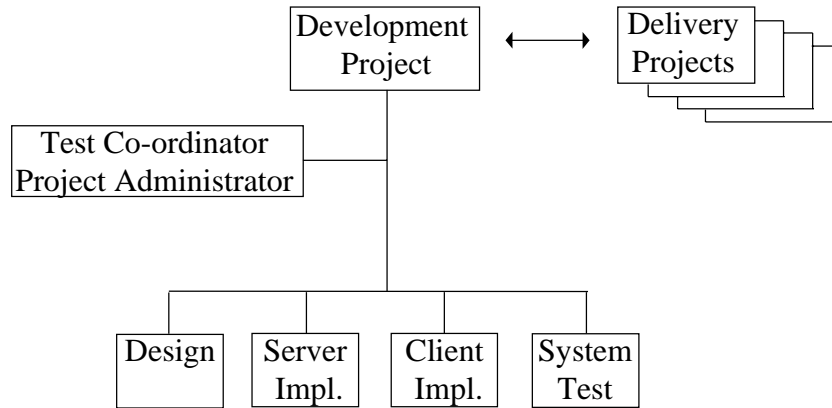
Scope

- **Server**
 - Standard HW, OS and dB application
 - Application 10 000 Kloc
- **Client**
 - Standard OS
 - HW based on standard components
 - Application 1 200 Kloc

Project Goals

- **Y2K**
- **Redesign of one sub-system**
- **Add old functionality to new customers**
- **Small improvements across the system**
- **New boot code and HW prototype**
- **Error corrections**

Organisational issues



© Enea Data AB A post-mortem analysis... 1.0 990919 Mats Grindal 5 (11)

ENE A DATA

Organisational issues

- **Integration test**
 - Own subproject
- **Test Co-ordinator**
 - Help optimise test efforts
- **Customer Projects**
 - Prevent unnecessary noise

© Enea Data AB A post-mortem analysis... 1.0 990919 Mats Grindal 6 (11)

ENE A DATA

Organisational issues

- **Resource Allocation**
 - Two different schemes
- **Role of System Test**
 - Quality assessments

© Enea Data AB A post-mortem analysis... 1.0 990919 Mats Grindal 7 (11)

ENE A DATA

Planning Phase

- **System Test Time budget**
 - Rule of thumb 15%
 - Budget result 10% (14%)
 - Final result 10.6 %

© Enea Data AB A post-mortem analysis... 1.0 990919 Mats Grindal 8 (11)

ENE A DATA

Preparation Phase

- **Time consumption**
 - 40% of system test time
- **Non-frozen requirements**
 - Prepare for changes
- **Quality of test cases**
 - Three areas of knowledge crucial
- **Inspection**
 - Needs to be planned

© Enea Data AB A post-mortem analysis... 1.0 990919 Mats Grindal 9 (11)

ENE DATA

Execution Phase

- **Hand-over meeting**
 - System test gains control
- **Regression strategy**
 - Three sets of tests
- **Non-functional testing**

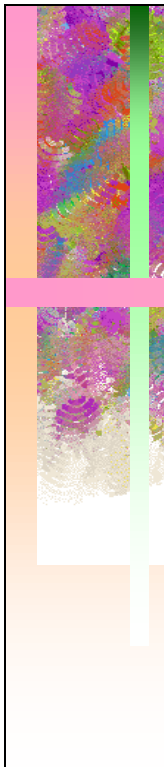
© Enea Data AB A post-mortem analysis... 1.0 990919 Mats Grindal 10 (11)

ENE DATA

Reporting Phase

- **Error Reporting Process**
 - Process and tool
- **Basis for delivery recommendation**
 - Metrics

Slide 1



A model for analysis of software testing metrics for process improvement

T Ashok(ash_t1@verifone.com)
Hema Gollamudi(hema_g1@verifone.com)
Piyali Biswas (piyali_b1@verifone.com)

VeriFone India Ltd., Bangalore, INDIA
A division of Hewlett-Packard

Slide 2



Agenda

- The motivation to analyze data
- The model of data analysis
- Metrics analysis
- Prediction model
- Conclusions
- References

Quality Week Europe '99

Slide 3

Motivation

- Performance improvement by improving processes
- The 'Goal-Question-Metrics' paradigm followed in identifying goals and metrics needed
- The metrics collected, analyzed are towards this end of satisfying the goals identified

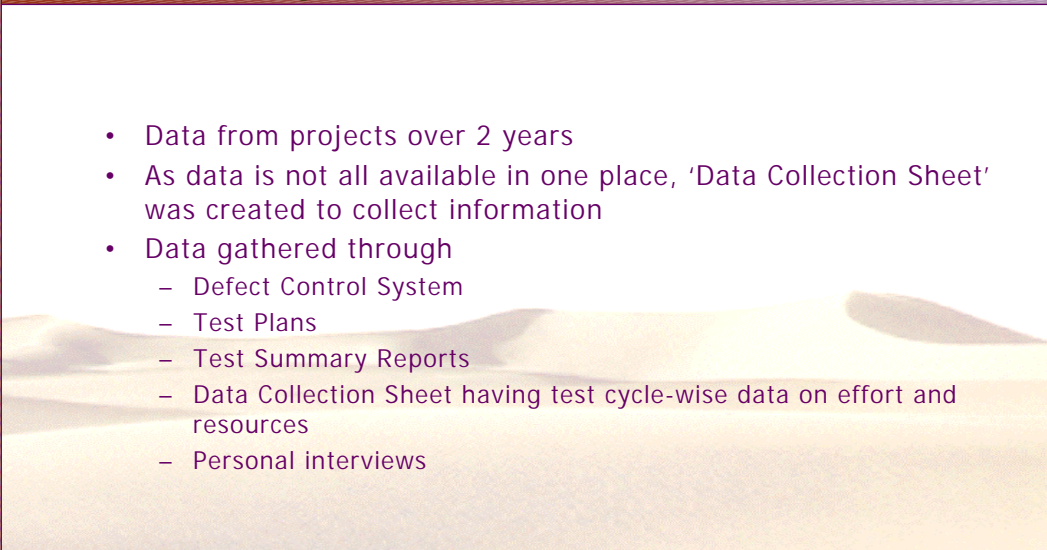


Quality Week Europe '99

Slide 4

Data analysis model

- Data from projects over 2 years
- As data is not all available in one place, 'Data Collection Sheet' was created to collect information
- Data gathered through
 - Defect Control System
 - Test Plans
 - Test Summary Reports
 - Data Collection Sheet having test cycle-wise data on effort and resources
 - Personal interviews



Quality Week Europe '99

... Data analysis model

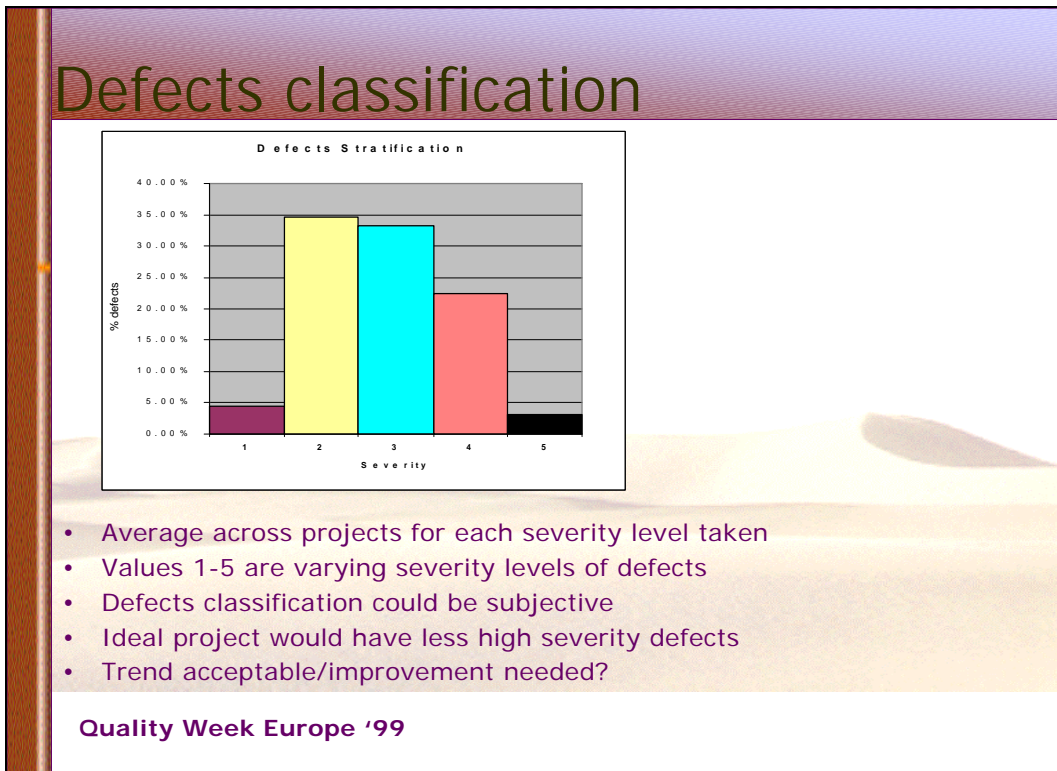
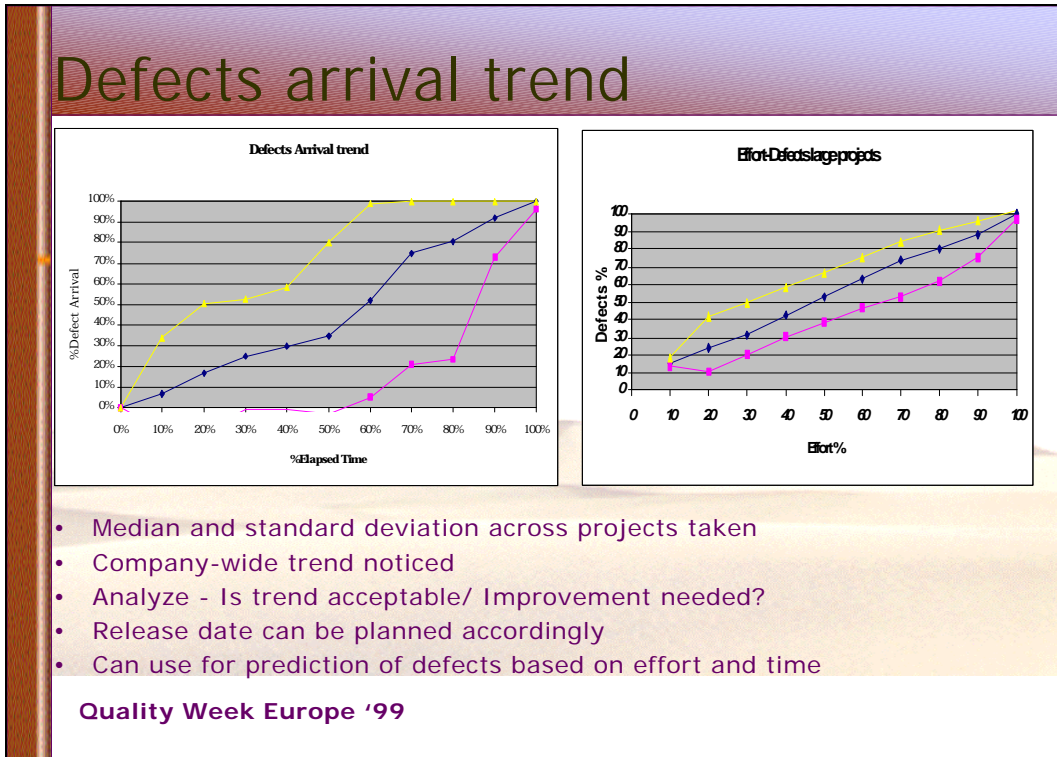
- Sheet is filled at start of project and to be updated regularly
- DCS is updated with each defect
- Test plans and Test Summary Report created at start and end of project
- Data from all sources collected and analyzed to give information
- In most cases trends are observed for metrics
- Process improvements are suggested

Quality Week Europe '99

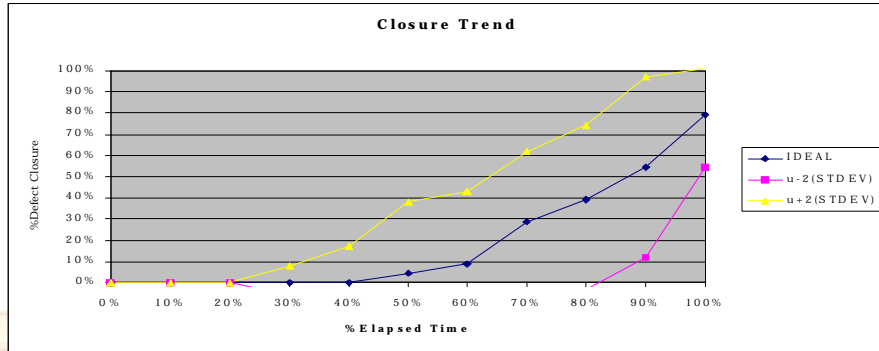
Some common terms

- **Normalization** of defects/effort/time : The total is taken as 100% and data at a point in time is taken as a percentage of the total
- **Mean/Median** of metrics taken by taking mean at each data point
- **Standard deviation** of metrics is taken at each data point
- **Large projects** have more than 5 testing cycles
- **Small projects** have less than 5 testing cycles
- Defects and time is always calculated from the start of system test phase
- For each of the metrics there are exception values which are discounted in calculating

Quality Week Europe '99



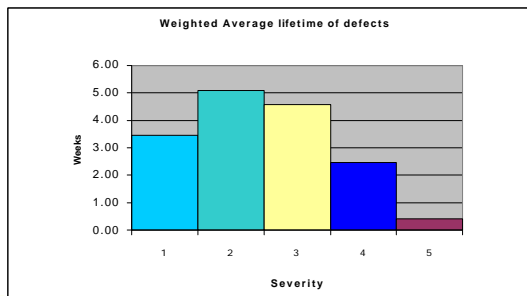
Defect repair rate



- Median and standard deviation across projects taken
- Trends noticed company-wide
- Prediction of open defects at release can be made
- Decision on releasing product on date can be made
- Trend acceptable/ Improvement needed ?

Quality Week Europe '99

Weighted lifetime of defects



- Weights assigned to varying severities
- Average across projects taken for each severity level
- Company-wide averages noted as baselines
- Are ranges acceptable / improvement needed
- Guidelines on maximum lifetime to be made

Quality Week Europe '99

Defects volumes/ Cost of defect

- Code size or function points can be used for defect volumes analysis
- U.S industry standards show direct correlation in code size and total defects
- Cost of a defect is total effort/total defects * labour cost
- Person hours per defect per project is plotted
- Range of Cost of defect is noted in control chart
- Ideal value for cost to be analyzed over projects
- Low value shows a poor quality product, effective testing
- High value could mean ineffective testing, good quality product

- Analyze no. of projects with extreme values in defect volumes
- Analyze range of values in a control chart
- Analyze overall and selected averages
- Baseline company-wide averages
- Study further projects for updates on baselines

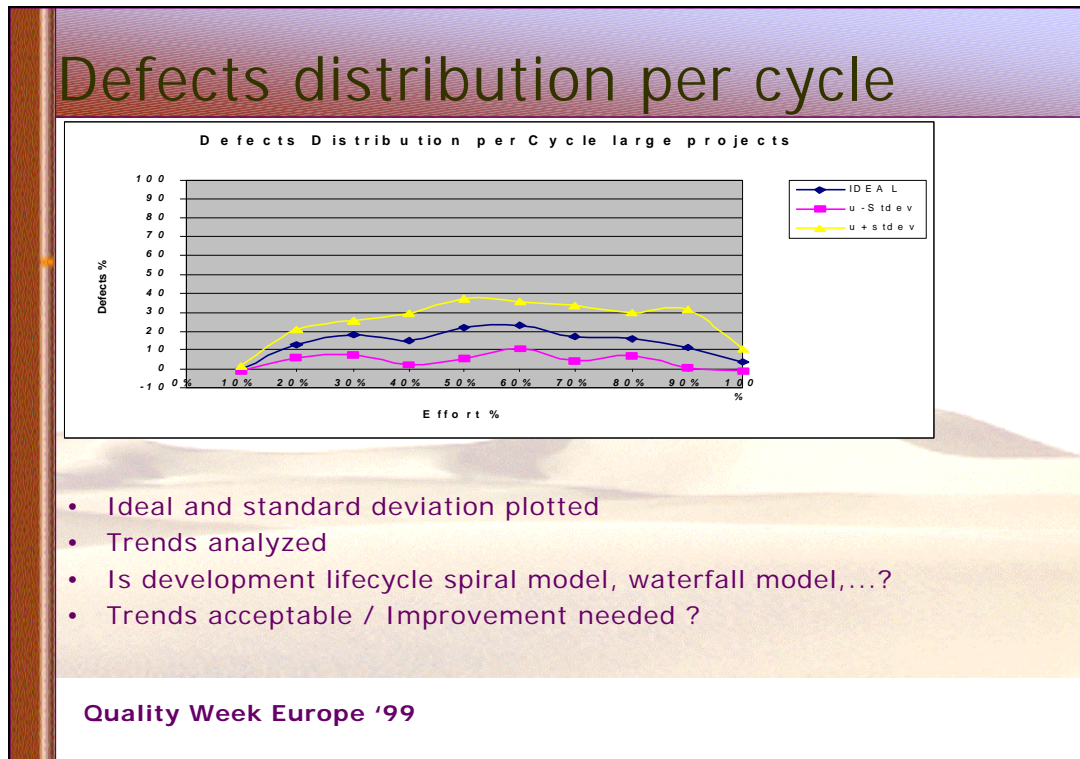
Quality Week Europe '99

Defect origins

| Origin | Defects % |
|--------|-----------|
| 1 | 85 |
| 1 | 70 |
| 1 | 45 |
| 1 | 35 |
| 1 | 25 |
| 1 | 15 |
| 1 | 10 |
| 1 | 5 |
| 2 | 85 |
| 2 | 55 |
| 2 | 40 |
| 2 | 5 |
| 3 | 15 |
| 3 | 5 |
| 4 | 25 |
| 4 | 20 |
| 4 | 10 |
| 4 | 5 |
| 5 | 55 |
| 5 | 45 |
| 5 | 35 |
| 5 | 25 |
| 5 | 10 |
| 5 | 5 |
| 6 | 90 |
| 6 | 75 |
| 6 | 65 |
| 6 | 55 |
| 6 | 45 |
| 6 | 35 |
| 6 | 25 |
| 6 | 15 |
| 6 | 10 |
| 6 | 5 |

- Defects are normalized
- Analyze reasons for maximum defect
- Improve processes in areas of maximum defects

Quality Week Europe '99



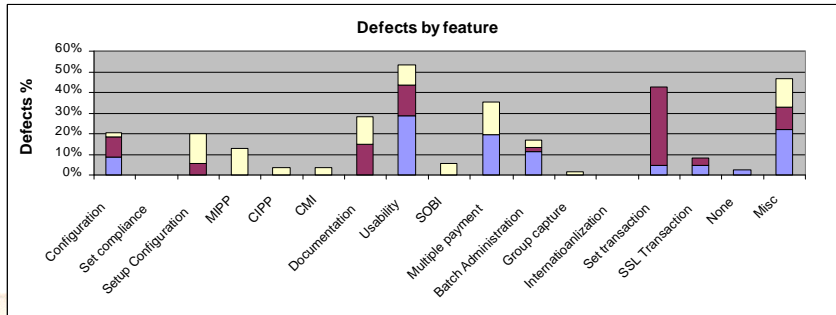
Effort Vs. Elapsed time

| Project | Elapsed time PH | Effort PH | Effort/Elapsed time |
|-----------|-----------------|-----------|---------------------|
| Project A | 2340 | 756 | 0.32 |
| Project B | 420 | 420 | 1.00 |
| Project C | 612 | 325 | 0.53 |
| Project D | 2232 | 1980 | 0.89 |
| Project E | 90 | 90 | 1.00 |
| Project F | 792 | 372 | 0.47 |
| Project G | 630 | 433 | 0.69 |
| Project H | 2534 | 1728 | 0.68 |

- Analyze the range of ratio, prepare control chart
- Analyze the average value
- Is the value acceptable / improvement needed?
- Have the schedules to be planned and monitored better?

Quality Week Europe '99

Feature wise defects distribution



- Analysis done for each product over various versions
- Defects are normalized
- Features with maximum defects in all versions observed
- Root cause analysis done for maximum defects
- Are high defects due to requirements/design/...?

Quality Week Europe '99

#Testcases Vs. Defects

| Total Testcases | No. of Defects | Ratio |
|-----------------|----------------|-------|
| 25 | 6 | 4.17 |
| 100 | 47 | 2.13 |
| 200 | 46 | 4.35 |
| 270 | 42 | 6.43 |
| 375 | 113 | 3.32 |
| 1235 | 66 | 18.71 |
| 2845 | 508 | 5.60 |

- Analyze the ratio of testcases to defects
- Analyze exceptions
- Prepare a company-wide baseline based on the average
- Analyze process improvements for improving the value
- Better test cases vs. better quality product

Quality Week Europe '99

Defects removal by severity

| Severity | >= 4 weeks | >=3 weeks | >=2 weeks | >=1 week | <=1 weeks |
|----------|------------|-----------|-----------|----------|-----------|
| 1 | 16.99% | 20.05% | 17.85% | 16.34% | 32.13% |
| 2 | 6.54% | 20.62% | 14.47% | 14.81% | 45.82% |
| 3 | 7.31% | 14.87% | 11.49% | 11.93% | 56.52% |
| 4 | 8.03% | 24.11% | 9.48% | 11.61% | 49.86% |
| 5 | 9.26% | 8.39% | 12.81% | 15.06% | 55.63% |

- Analyze defects removal intervals for each defect severity
- Are the values acceptable?
- Are appropriate response times for varying severities followed
- Process improvement needed for improving data?

Quality Week Europe '99

Prediction model

- Based on code size, defects in code can be predicted
- Based on effort estimated, the total testing time can be predicted
- Based on effort estimated, total defects can be estimated
- Based on total defects estimated, total open defects at end of testing can be predicted
- Based on total open defects, release date can be predicted

Quality Week Europe '99

Conclusions

- Individual projects give variant data. Normalize across projects
- Arrival and closure rates of defects are analyzed and improved
- Utilization of time and effort are explained and improved
- Company- wide baselines are created and improved
- Predictions on behaviour of projects in the SDLC model can be made
- Main process improvement areas are noted and worked on

Quality Week Europe '99

References

- Applying Software Metrics by Paul Oman and Shari Lawrence Pfleeger
- Applying Software Measurement: Assuring Productivity and Quality by Capers Jones
- Software Metrics: A rigorous approach by Norman A Fenton
- Metrics and Models in Software Quality Engineering by Stephan H. Kan
- Software Metrics: Establishing A Company-Wide Program by Grady and Caswell

Quality Week Europe '99

Error Trending

Why and How

Niels Bruun Svendsen
B-K Medical A/S, Denmark
nbs@bkmed.dk

Introduction

How do you waste your money? Do you make the perfect error free product and loose the market while doing so or do you get your product out "first thing" and drown in error corrections, patches and possibly field updates?

When developing systems and software an inevitable management question is: "When is the system ready for release?". On the bottom line the answer on when to release a new product for production and sales is a matter of being able to estimate the cost of releasing, as well as the cost of postponing the release.

In calculation of the cost of releasing a product the number of remaining unknown errors is a major factor. In that respect error detection trends during the system-testing phase are interesting as means of estimating the number of remaining unknown errors. This paper will share the experiences gained and the lessons learned from introducing error trending as an estimation tool and highlight the benefits found as well as the problems encountered.

The results includes not only experiences with the precision of the estimates but also, and not less interesting, the impact of error trending on the organization. It was found that the error trend had a great value during all of the system-testing phase, and for all groups involved:

- Top-management get a more objective estimate on remaining unknown errors
- Project managers gets a management and planning tool and arguments (against sales and top-management) for not being able to release "tomorrow"
- System test staff gets a planning tool and the arguments for extra resources
- The developers get the argument for not being able to start on another project "tomorrow", i.e.: "When this many errors are suspected to be found, we need time to fix them!"

In short this means that one simple curve gives input and insight for top-management, project management, QA function and developers, i.e. becomes the common reference on system state.

Company Context

B-K Medical develops, produces and markets ultrasound systems for medical diagnostic imaging. The systems are sold throughout the world with the major markets being Europe, USA and Asia. B-K Medical has 250 employees with 166 located in Denmark. The development department consists of 60 employees where 20 are involved in software development. B-K Medical is ISO 9001 certified and most of the products have FDA market clearance and are CE-Medical Device certified. Therefore external audits are performed accordingly. No formal assessment against a model has been performed, but an informal self-assessment using the BootCheck tool from ESI has been performed. This assessment gave maturity ratings between 2.5 and 3.25, indicating some areas in need of improvement to get to the Defined (3) level, and a general lack of metrics as required in the Managed (4) level.



Project Initiation

The introduction of error trending at B-K was initiated by a management request for an improved basis for making the release decision, i.e. to decide whether or not to release a new product for production and sales. As part of the initiatives taken in order to pursue this goal, Error Trending was introduced. By using Error Trending to estimate the number of remaining unknown errors rather than using pure intuition, the objectivity of the basis for the release decision is increased.

Although aiming primarily on an estimate of remaining errors at the time of the release decision, error trending was introduced in the system-testing group as a tool to be used from the beginning of system test execution until the product is released. Beginning error trending early in the system-testing phase gave a lot of good experiences as described later in this paper.

The initial steps with error trending were done on error data from a scanner that had been on the market for a year and therefore the number of reported error after release was known. Error reports from the last part of the system-testing phase were used and plotted as seen in fig.1. The y-axis shows the accumulated number of errors reported, and the x-axis shows the number of test days. A test day is equal to a calendar day except that only calendar days where test were performed are included.

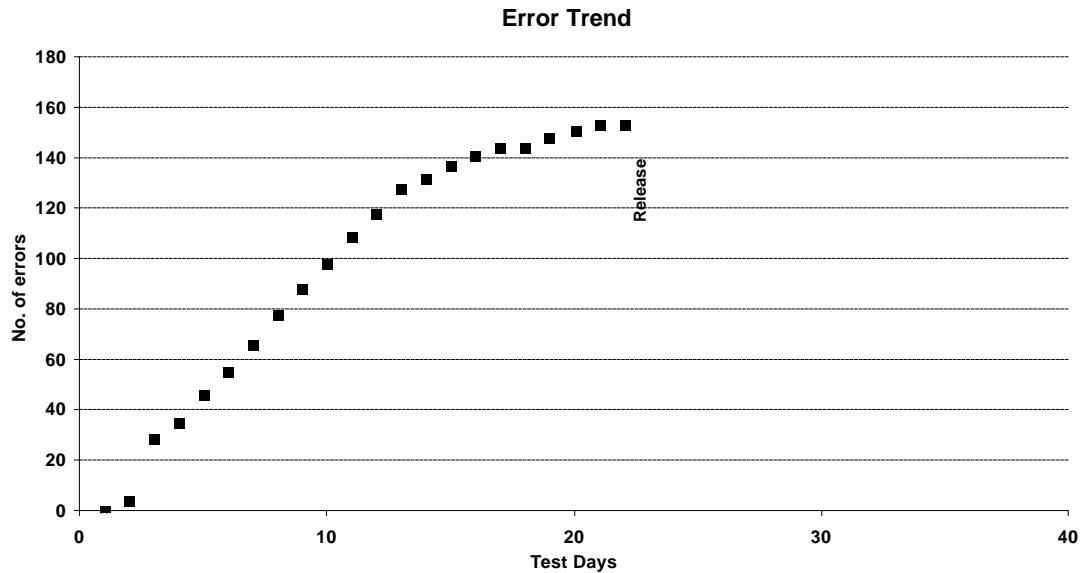


Fig.1 : Accumulated no. of errors for released product

Despite the fact that the test effort pr. test day was not known in any great detail, the plotted error data gave a quite clear trend with a distinct convergence in the last part of the trend. To start off with, very simple functions were tried out, using the trending functions in the MS Excel spreadsheet. None of the experiments using all data gave any trustworthy results. Our criterion for a result to be trustworthy, were that the estimated trend had a good correlation with the last converging part of the data, and that it gave an estimated total number of errors higher than the number of errors already found.

Finally it was decided to focus only on the latter part of the error data, and using the exponential function on those data as shown on fig. 2. It gave a perfect match with the number of errors actually found after release.

Although this was very well affected by the fact that we knew the result we should get, it did give some confidence in that here was something useful. Fig. 2 was used for raising internal interest in error trending, with the argument that:

Based on data with a great deal of uncertainty you can apparently still draw and extrapolate a trend using the data from the final stage of system test and get a very good estimate on remaining errors.

The conclusions on the work with the data from the released product was that although limited in amount and precision it gave a good initial interest in error trending and was a kick-off for going further into the subject.

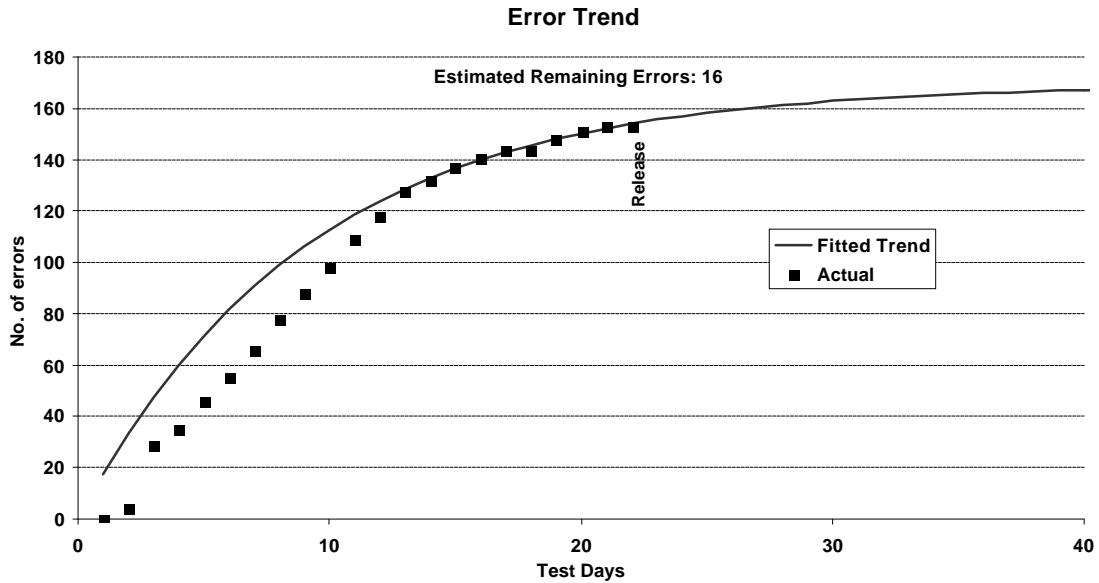


Fig. 2 : Exponential Error Trend for released product

The Model

When searching for experiences on error trending the name of SATC (Software Assurance Technology Center) at NASA is very likely to pop up. SATC has published articles that mentions their work on an Error Trending Model, ref.[1] & ref.[2]. The Error Trending model was also mentioned by Linda Rosenberg, SATC at a QWE'98 tutorial. As we did not find any further description of this model, Linda Rosenberg was contacted. We got a very quick response saying that work was still in progress on the model and they were working on a tool to support the model. We were also invited to send our data to SATC to have them analyzed.

We decided to send data from the first part of system testing on a new scanner. They returned a spreadsheet with our data analyzed by a Weibull variate. This was actually not the model used by the tool, but our data performed better in this model. It differed slightly from the Weibull function in relation to the manpower utilization, but as this did not influence the estimate on the number of remaining errors, we decided to proceed with a model based on the Weibull function itself. The 2-parameter Weibull function has the form:

$$f(t) = K * \left[1 - \exp \left[- \left(\frac{t}{t_{\max}} \right)^p \right] \right]$$

where t is the test effort, p is the shape parameter, t_{\max} is the scale parameter and K is the scale-up constant (total number of estimated errors).

With $p = 1$, we have the exponential function and with $p = 2$, we have the Rayleigh curve. When used for trending, the parameters K , t_{\max} and p are optimized to get the minimum sum-of-difference-squared. The spreadsheet included set-up for using the MS Excel solver to analyze additional data, and has formed the basis of our further work with error trending. We are therefore very thankful for this valuable input from SATC.

In fig. 3 the use of the Weibull function on the data from the released scanner is shown. The estimated number of errors remaining is 5. A total of 15 error reports have been made since release, including also change request, so although a bit low, it is still a good estimate, based on data with some uncertainty.

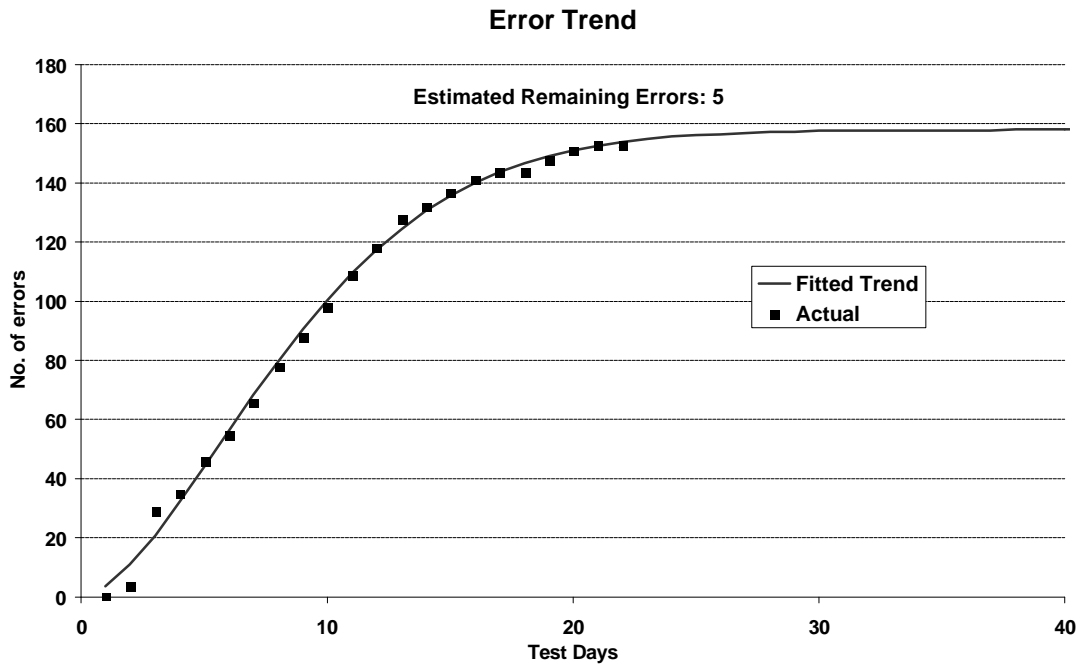


Fig. 3 : Weibull Error Trend for released product

Error Trending during System Test

As mentioned earlier, data from the first part of system test on a new scanner were analyzed by SATC, NASA. The results, based on the Weibull function, gave a very high estimate on the number of remaining errors, as well as a high number of days to find the remaining errors.

When presented for the project manager we had the first direct impact on the project:

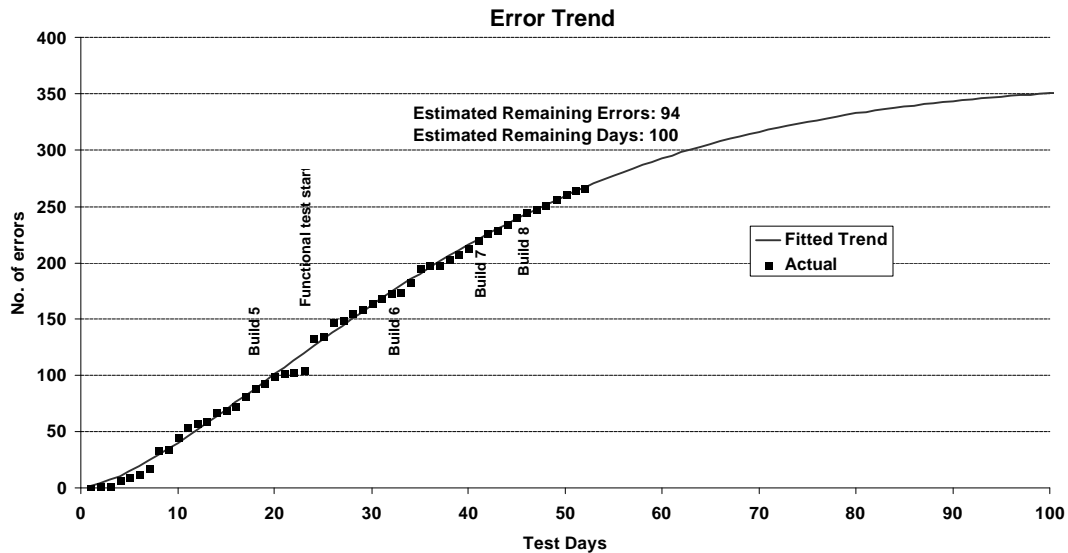
With that many test days left, we need more test objects

The presented error trend and estimates were the direct cause for additional test objects

to be arranged for. The fact that the calculations on our data were made by NASA was used to increase confidence in the estimate.

A good reference gives confidence

From this point in the system test phase, daily updates of the trend and estimates were made, i.e. yesterdays reported errors were entered and new parameters for the Weibull function were calculated. The test days are here counted as test man-days, Fig.



4 : Weibull Error Trend for new product

e.g. 3 testers working one day, results in 3 test days. This way we account for the changes in test effort.

The new trend and estimates were presented on the “project wall”, and on the Intranet, see fig. 4. A lot of internal interest were gained and although not all understood that the error trend curve were optimized every day, it gave opportunities to discuss the state of system under test as well as error trending in general.

During the last part of the system test phase the project manager had a demonstration of the system for the top-management. A full functioning scanner was demonstrated and as often in these situations the comment that the project manager receives is: “This scanner looks complete. Why don’t we release tomorrow or at least at the end of the week?”. The standard answer to this question is that “we still need a little optimization on the quality of the image” and “we haven’t got all parts in production quantities”. But this time the project manager had another argument, i.e. the error trend and the estimate of remaining errors and test days. So he showed the error trend saying: “See we estimate the need for another 100 test days. With the number of test objects and testers we have, that means we’re finished in 30 days, and that is exactly the planned release date. That was very convincing and made the end of that discussion. Of cause the input from SATC at NASA again played a role in the creation of confidence in the estimate.

*The product is finished. Why not release “tomorrow”?
The Error Trend holds the answer*

This time it was the top-management, but next time it could be the sales staff asking for a release “tomorrow”. The visualization of the Error Trend makes it easy to communicate the probability for further errors to all types of staff in the company.

Not only the project manager, but also the developers can make use of the Error Trend in this stage of the project. Typically another project is crying out for development resources as soon as they have finished their work on the current project. And there is a strong tendency for developers to be almost finished, i.e. “I have only a few more (known) errors to correct, then I’m finished”. Here the Error Trend is a great help too as it is easy to take the number of estimated remaining errors and divide by the number of developers and you have an estimate on how many more errors there are to correct for each developer. In our case and probably for many others, this will mean a considerable amount of time to be planned for before the resources are ready for the next project.

*You have implemented it all. Why can't you start on a new project
“tomorrow”?
The Error Trend holds the answer*

The value of the Error Trend and the estimates in the mentioned situations naturally depends on the precision of the estimates. However we find that the normal expectations are that far from any reality, that almost any estimate is better than none. The benefit is there if just you can show that there is “a lot” of errors left and not just “a few”.

Fluctuations of the Error Trend

Fluctuations in the trend was expected, as new builds, new test techniques and the start of test in previously untested areas are very likely to initially increase the number of errors found. And when you test on the same build with the same test technique fewer and fewer errors will be found. This phenomenon is seen in fig. 4, where the first 24 test days constitutes its own “S” curve and a large increase in error detection rate is seen as we enter what is referred to as functional test.

So fluctuations are seen:

- When test of new features is started
- When changing test techniques
- When new builds are introduced

In our case the largest fluctuations were seen when entering test of new feature and the smallest fluctuation seen when introducing new builds.

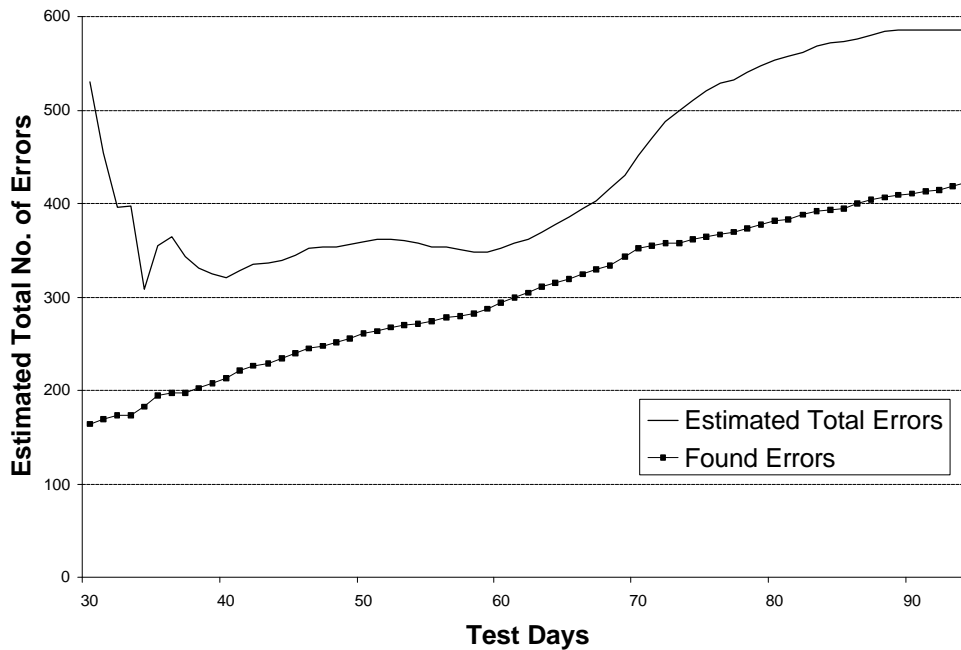


Fig. 5 : Evolvement of estimated total no. of errors

As the estimated total number of errors were calculated every day these fluctuations had an impact on the estimated total number of errors. Therefore there was a need for visualization of the evolvement of this estimate. A trend for the estimated total number of errors was added as seen in fig. 5. The first estimate of 530 errors in total was the estimate received from SATC’s analysis of our data and the figure used to get additional test objects. As seen the estimate was reduced somewhat during the first period where Error Trending was used and we saw the estimate stabilize around approx. 350 errors. But then around the 65th test day suddenly the estimates of the total number of errors increased drastically. This was caused by the fact that we had entered test of 2 previously untested areas that were found to have a much higher error density than what had been tested so far.

This increase in the estimated number of errors in the system naturally imposed a problem on the project, both in getting development resources to correct the errors and the extra time needed for both the correction and the verification of the corrections. When discussing the situation we could see that this was not a new problem, but rather a problem many projects has suffered from. It is a result of the way we plan the system test, where we execute the test sequentially, function by function. The problem is visualized in fig.6. The illustration shows a set of functionalities, where the “F” functionality is significantly more error prone than the others. The first case

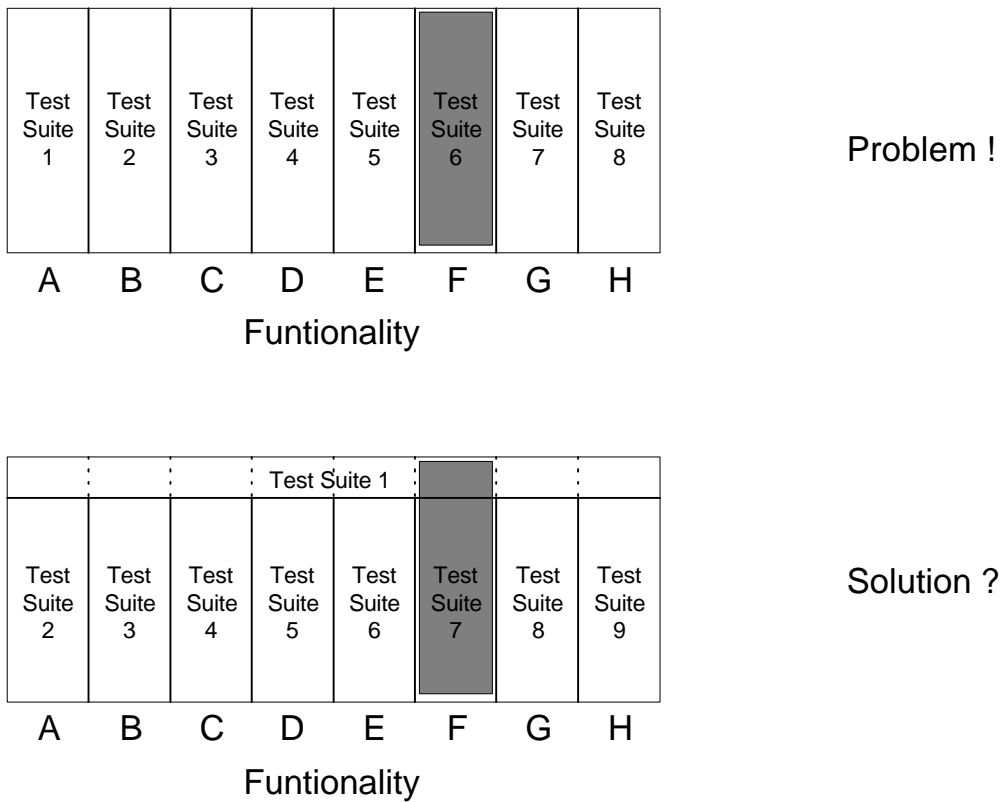


Fig. 6 : Test Sequences

is how we traditionally have covered the test of such a system with test suites for each functionality and executing the test suites sequentially. This means that we will not have any knowledge of the, in this case, high error density of “F” until late in the system test execution phase.

Therefore we have changed the strategy for test planning slightly, making a test suite that covers all functionalities. This test suite will not cover any functionality in depth, but just enough to get an impression of the error density of the functionality. Use Cases will be used for designing this test suite. By executing the Use Case based test suite as the first test suite, we will get valuable data for planning the execution of the remaining test suites. We will also get the possibility to reject functionalities early in the test process, limiting the time spend on system testing features that are not ready for system test. This way of planning the execution of the system test will be applied in two new upcoming projects.

Common Sense has to be triggered

This change in the system test execution is not directly connected to the Error Trending. But the visualization of the problem that the Error Trend caused was the trigger needed to realize it and to have a broader group of people discussing the problem and possible solutions.

In the final stage of the system test it was found that the estimated total number of errors remained at a very high level, even with many test days having no or very few errors found. When looking at the trend curve it was apparent that it was not following

the actual error findings in the final stage of the system test very well. Therefore the initial part of the system test, where new features were still added to the system, was omitted from the trend calculations in the final stage of the system test. The lesson learned here is that:

- Error Trending had valuable impact from early in the system test, even though not all of the system was ready
- Estimates to be used in the latter part of system test had to be based solely on data starting at the time where the total system is available.

Summary

The experiences with the work performed with Error Trending can be summarized in the following Why's and How's:

Why:

- It triggers the use of common sense
 - It highlights the need for process improvements
- It's a valuable tool implemented by simple means
- It works as a common reference on system state
 - Managers gets valuable knowledge
 - Developers gets valuable knowledge
 - Testers gets valuable knowledge
- It improves the release decision support

How:

- Find and plot available data
- Select an error trend approach
 - Ideas can e.g. be found in literature by SATC, J.D.Musa, Grove Consultants and S.H.Kan
- Apply it during system test
 - Make it visible to all involved in the project
 - Monitor the trend and learn from the questions and discussions it generates
- Do not initially expect high-precision estimates
 - Increase accuracy by process improvement actions

Conclusions

We started off aiming at a technique to estimate the number of remaining errors at the time of possible release. What we found was a technique that apart from doing that were able to trigger common sense in several processes related to the system test phase. Improvement was triggered in relation to:

- Being realistic about when development resources are ready for the next project
- Getting an easily communicable argument for not releasing “tomorrow”
- Planning the system test for early error density overview

So far we have limited data on the precision we can obtain, but we have found that even with a limited precision there’s a lot of benefit in collecting and presenting data which in many cases are fairly easy to get hold of.

Apart from the mentioned models we also tried using 3rd order polynomial approximation as suggested by Grove, but had some problems getting estimates we believe in. And the trust in the model is a key issue when the idea is to be “sold” internally. Also a good reference play a key role too in that respect. But whatever model you choose, don’t trust it blindly. Keep your common sense and professional knowledge, but let Error Trending help you stay objective and use it as a mean of communicating between personnel groups.

Get your Error Trending started – You won’t regret it

References

- [1] Robert E. Waterman, Lawrence E. Hyatt : ”Testing - When Do I Stop?”
International Testing and Evaluation Conference, Washington, DC - October, 1994
- [2] Dr. Linda Rosenberg, Ted Hammer, Jack Shaw: “Software Metrics and Reliability”
9th International Symposium on Software Reliability Engineering Germany - Nov 1998
- [3] Stephen H. Kan: “Metrics and Models in Software Quality Engineering”,
Addison Wesley, ISBN 0-201-63339-6

Error Trending

Why and How



Niels Bruun Svendsen
B-K Medical A/S
email: nbs@bkmed.dk



ultrasound
we specialize because you do

Error Trending, Background

- **How do you waste your money?**
 - Release too early ⇒ error corrections
 - Release too late ⇒ loose market share
- **Support for the release decision needed**
 - How many unknown errors are left?
 -

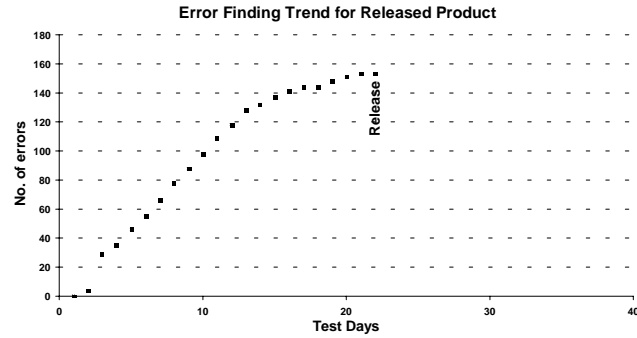


ultrasound
we specialize because you do



Error Trending, Initiation

- Analyzing available data
 - Plotting the data gives overview

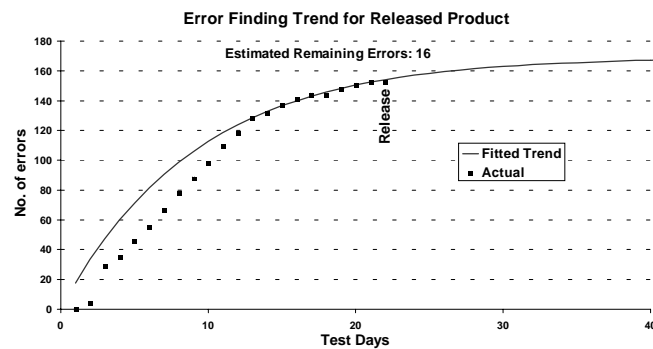


ultrasound
we specialize because you do



Error Trending, Initiation

- Analyzing the available data
 - Adding a trend that looks right gives interest



ultrasound
we specialize because you do

Error Trending, Initiation

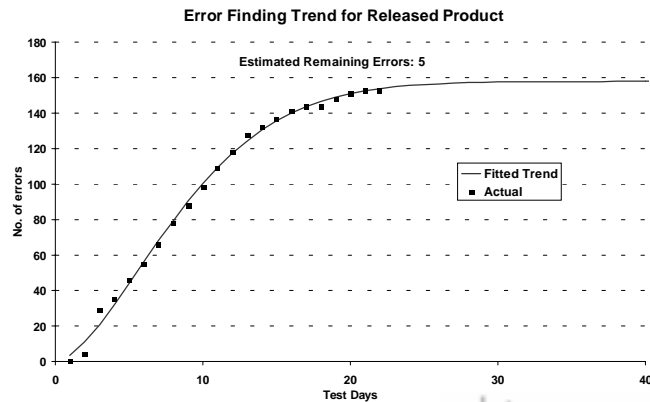
- **SATC, NASA**
 - Waterman error trending model
 - New tool called SETT, SW Error Trending Tool
 - Analysis on our data
 - Suggested a Weibull variate
 - Returned spreadsheet with our data
- **Proceeded with Weibull**



ultrasound
we specialize because you do

Error Trending, Initiation

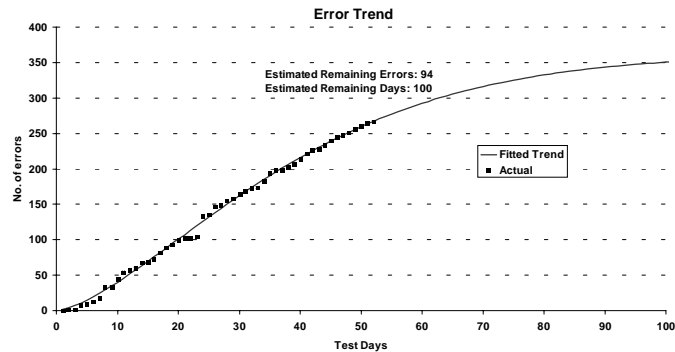
- **Analyzing the available data**
 - The Weibull model gave confidence



ultrasound
we specialize because you do

Error Trending, in system test

- Analyzing data during system test
 - Using the Error Trend as common reference on system state



B-K Medical

ultrasound
we specialize because you do

Error Trending, in system test

- Testers:
 - With that many errors left we need another test object
- Project Manager:
 - With that many errors left we do not release “tomorrow”
- Developers:
 - With that many errors left we do not start on a new project “tomorrow”

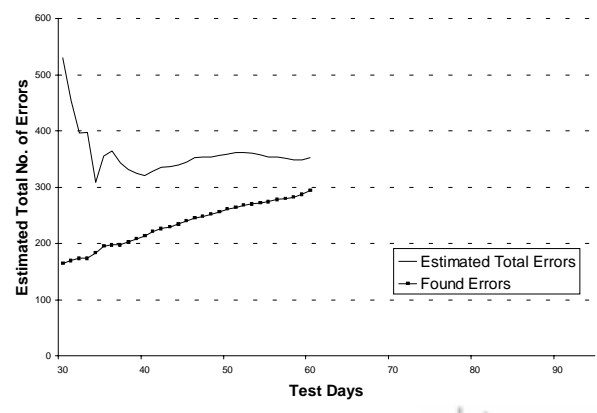
B-K Medical

ultrasound
we specialize because you do



Error Trending, in system test

- Analyzing data during system test
 - Evolvement of the Error Estimate

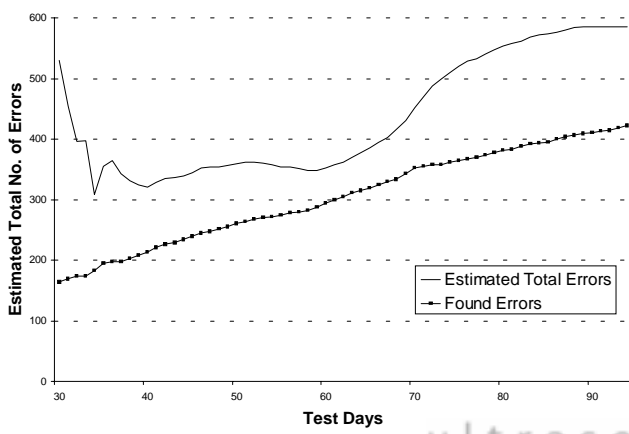


ultrasound
we specialize because you do



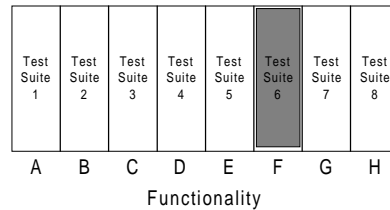
Error Trending, in system test

- Analyzing data during system test
 - Evolvement of the Error Estimate

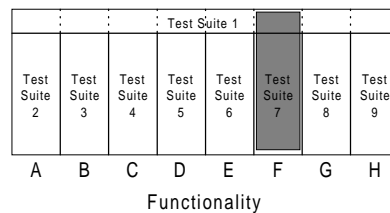


ultrasound
we specialize because you do

Error Trending, Test sequence



Problem !



Solution ?

B-K Medical

ultrasound
we specialize because you do

Error Trending, Conclusions

Why:

- **Trigger use of common sense**
 - Highlights need for process improvements
- **Valuable tool by simple means**
- **Common reference on system state**
 - Managers gets valuable knowledge
 - Developers gets valuable knowledge
 - Testers gets valuable knowledge
- **Improves release decision support**

B-K Medical

ultrasound
we specialize because you do

Error Trending, Conclusions

How:

- **Find and plot available data**
- **Select an error trend approach**
 - SATC, J.D.Musa, Grove, S.H.Kan
- **Apply it during system test**
 - Make it visible to all involved in the project
 - Monitor the trend and learn from the questions and discussions it generates
- **Do not expect high-precision estimates**
 - Increase accuracy by process improvement actions



ultrasound
we specialize because you do

Slide 1



How we Implemented a Change Control Process

QWE99-v3.ppt

Slide 2

Sanlam

How we Implemented a Change Control Process

Information about the company

- Head Office in Cape Town South Africa
- 2nd largest supplier of Assurance Products
- 81 years old
- Computer systems up to 30 year old
- Many different Information Technologies

Company Culture

- Conservative
- Norming
- Creativity being
- Level 5 Maturity
- Demutualised
- Unbundled

Software Engineering Programme

- Corporate project
- Focus on processes
- Use of International Best Practice
 - CMM
 - ISO 15 504
- My focus on Quality
- Change control - a key issue



QWE99-v3.ppt

Slide 3

Transition Management

Why Transition is so Difficult

What you see

The hidden, total effect of past experience

Phases in Transition

- UnFreezing
 - Disconfirmation
 - Survival Anxiety
 - Safety
- Transition
 - Cognitive redefinition
 - Imitation of a role model
 - Scanning
- Stabllising
 - Congruence

Phases in Transition

UnFreezing Transition Stabllising

Emotional Stages

Loss
Shock
Disbelief
Sadness & Loneliness
Anger

Hopeful Optimism

Despair

Bargaining

Depression

Acknowledgement

Control Regained

Renewal

Creativity

New Self Knowledge

Relief

Acceptance

QWE99-v3.ppt

Slide 4

Transition Management

Planning the Transition

Pressure

Space

Empathy

Time

Managing Expectations

Change

What you wish for

What you plan for

What you get

Time

Three Requirements for Transition

- Reason to Move
 - Audit
 - History
- Vision
 - Action Research
- First Steps
 - Process
 - People
 - Project

QWE99-v3.ppt

Slide 5

Building the Vision

Action Research

Action Research - cycle 1

- Contracting - the gauntlet
- Interviews - anybody and everybody
- Digestion - endless discussion
- Feedback - the day of the bun-fight
- Actions - Processes, People, Project

Action Research - cycle 2

- Contracting - the false starts
- Interviews - the executives
- Digestions - more endless discussion
- Feedback - the slick sell
- Actions - OK guys now go and do it!

QWE99-v3.ppt

Slide 6

First Steps

The Process - Production Change Control

Processes

Implementation

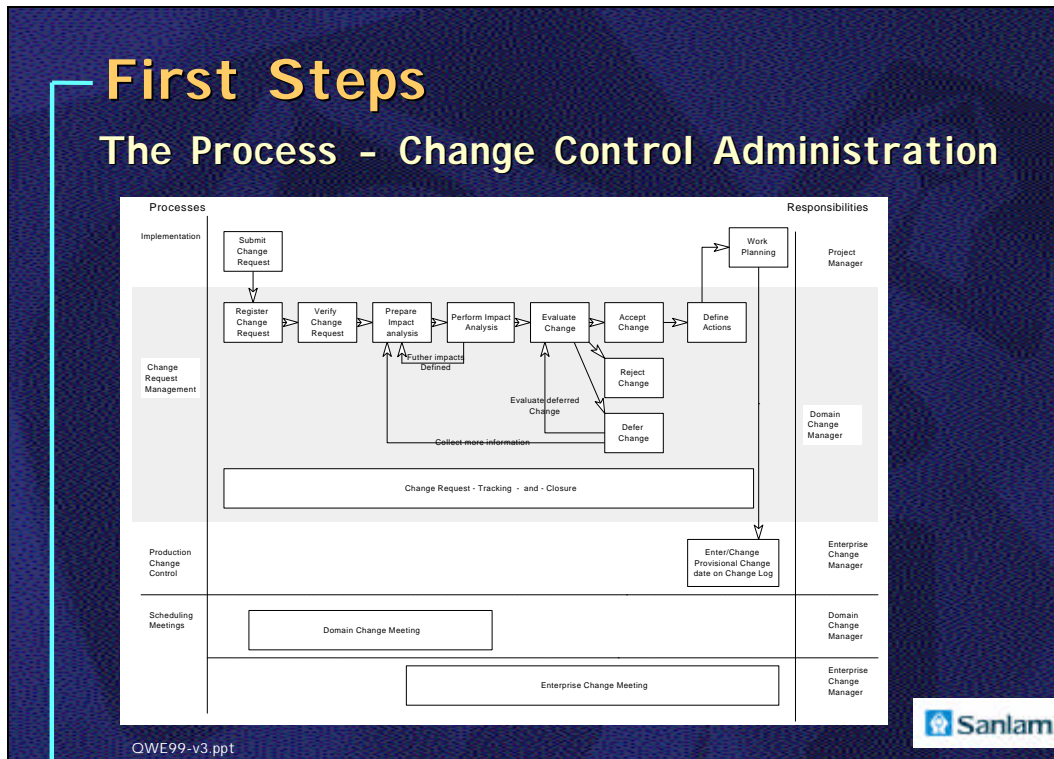
Production Change Control

Scheduling Meetings

Enterprise Change Meeting

QWE99-v3.ppt

Slide 7



Slide 8

- # First Steps
- People - Organisation**

 - Central Structure
 - Enterprise Change Manager
 - Decentralised Structure
 - Team Change Manager

Project

 - Phases
 - Environment - Scope
 - Processes
 - Tools
 - Implementation
 - What the project delivered
 - Processes
 - Organisation
 - Administration tool
 - Implementation sub-projects
 - Clarity and Momentum
- Sanlam
- QWE99-v3.ppt

Slide 9

Lessons

- **People - the key roles**
 - The Evangelist
 - Sets the scene
 - Creates the energy
 - Pushes the processes
 - The Project Manager
 - Admin and documentation
 - Co-ordination
 - The Enterprise Change Manager
 - Continuity
 - Foundation
- **Critical Requirements**
 - Sponsorship
 - Shared Vision
 - Shared Mental Models
 - No surprises in the project board meetings
 - Communication - don't rely on e-mail
 - Discussion crucial for shared mental models
 - Persistence
 - Victory and Defeat
- **Other Lessons**
 - Parallel developments are OK
 - People are an exponential factor
 - A thing worth doing is worth doing..?
 - You *MUST* have Panache

QWE99-v3.ppt




Slide 10

Transition Management is *VITAL*

So, have you...

- Defined the Change?
- Assessed the climate (to remove barriers)?
- Identified your approach to change?
- Identified the sponsors for the change?
- Prepared the personnel for the change?
- Assessed the culture?
- Developed the change agents?
- Established a motivation plan?
- Established a communication plan?
- Established an implementation plan?





How we Implemented a Change Control Process

QWE99-v3.ppt

Sanlam

How we Implemented a Change Control Process

Information about the company

- Head Office in Cape Town South Africa
- 2nd largest supplier of Assurance Products
- 81 years old
- Computer systems up to 30 year old
- Many different Information Technologies

Company Culture

- Conservative
- Norming
- ~~Norming~~ culture being
- ~~ISO 15504~~ Maturity
- Demutualised
- Unbundled

Software Engineering Programme

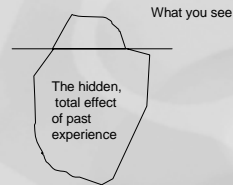
- Corporate project
- Focus on processes
- Use of International Best Practice
 - CMM
 - ISO 15 504
- My focus on Quality
- Change control - a key issue

QWE99-v3.ppt



Transition Management

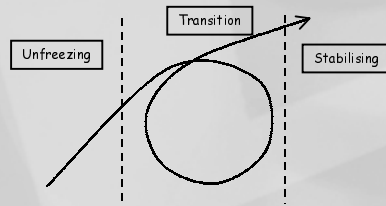
Why Transition is so Difficult



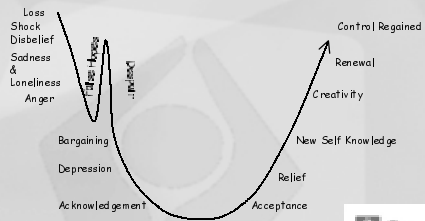
Phases in Transition

- UnFreezing
 - Disconfirmation
 - Survival Anxiety
 - Safety
- Transition
 - Cognitive redefinition
 - Imitation of a role model
 - Scanning
- Stabilising
 - Congruence

Phases in Transition



Emotional Stages

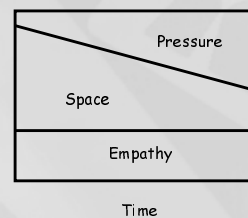


QWE99-v3.ppt

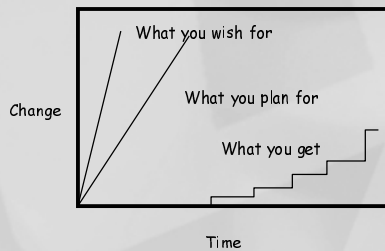


Transition Management

Planning the Transition



Managing Expectations



Three Requirements for Transition

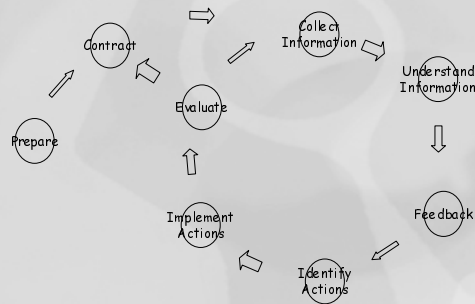
- Reason to Move
 - Audit
 - History
- Vision
 - Action Research
- First Steps
 - Process
 - People
 - Project

QWE99-v3.ppt



Building the Vision

Action Research



Action Research - cycle 1

- Contracting - the gaunt let
- Interviews - anybody and everybody
- Digestion - endless discussion
- Feedback - the day of the bun-fight
- Actions - Processes, People, Project

Action Research - cycle 2

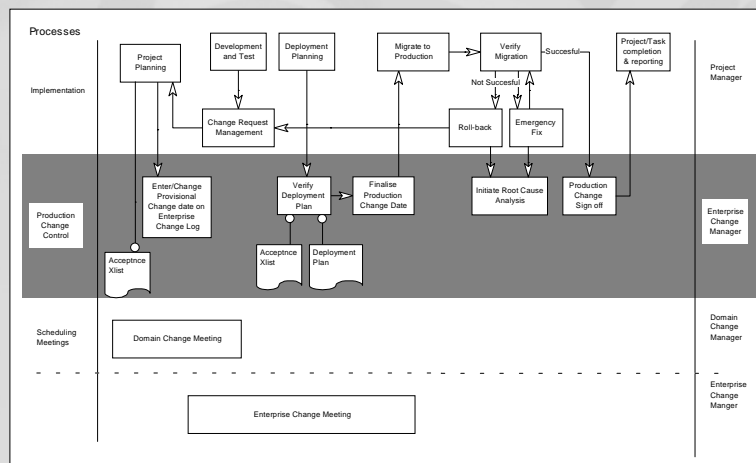
- Contracting - the false starts
- Interviews - the executives
- Digestions - more endless discussion
- Feedback - the slick sell
- Actions - OK guys now go and do it!

QWE99-v3.ppt



First Steps

The Process - Production Change Control

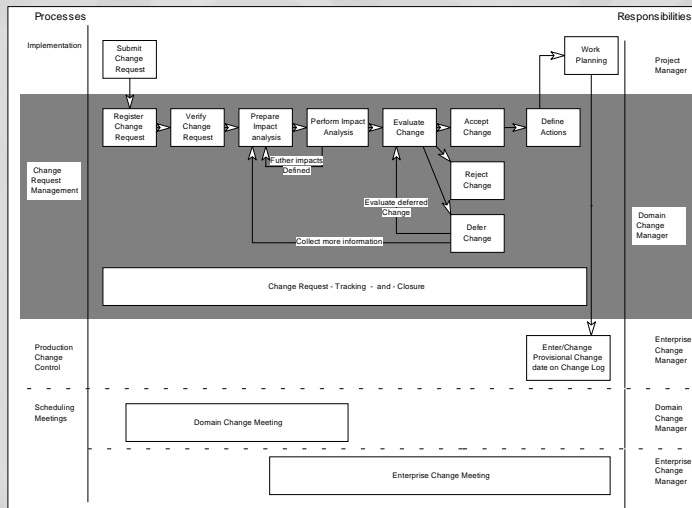


QWE99-v3.ppt



First Steps

The Process - Change Control Administration



QWE99-v3.ppt



First Steps

People - Organisation

- Central Structure
 - Enterprise Change Manager
- Decentralised Structure
 - Team Change Manager

Project

- Phase s
 - Environment - Scope
 - Processes
 - Tools
 - Implementation
- What the project delivered
 - Processes
 - Organisation
 - Administration tool
 - Implementation sub-projects
 - Clarity and Momentum

QWE99-v3.ppt



Lessons

People - the key roles

- The Evangelist
 - Sets the scene
 - Creates the energy
 - Pushes the processes
- The Project Manager
 - Admin and documentation
 - Co-ordination
- The Enterprise Change Manager
 - Continuity
 - Foundation

Other Lessons

- Parallel developments are OK
- People are an exponential factor
- A thing worth doing is worth doing...?
- You *MUST* have Panache

Critical Requirements

- Sponsorship
- Shared Vision
- Shared Mental Models
 - No surprises in the project board meetings
- Communication - don't rely on e-mail
 - Discussion crucial for shared mental models
- Persistence
 - Victory and Defeat



QWE99-v3.ppt

Transition Management is *VITAL*

So, have you...

- Defined the Change?
- Assessed the climate (to remove barriers)?
- Identified your approach to change?
- Identified the sponsors for the change?
- Prepared the personnel for the change?
- Assessed the culture?
- Developed the change agents?
- Established a motivation plan?
- Established a communication plan?
- Established an implementation plan?



Clean Management begins with Clean Applications

Kishor Bapat

EWO Software Inc.

Clean Management - Definition

- A practice of managing the hardware, operating system, applications, and network components to ensure once a system has been fully tested, future changes in any system component are also analyzed and tested as appropriate. "

Clean Management Corollary

- A practice of ensuring that applications contain *all and only* the components needed for them to function and to be properly maintained.
- All the components will be correctly identified and their inter-relationships will be correctly documented and maintained.

Clean Management

- Every Data Center needs to follow clean management practices.
- For too long, applications have been poorly constructed and maintained.
- Y2K and Euro have reinforced the need.
- Application change need not be the nightmare it currently is.

Clean Management - Basics

- Not a new concept
- Common Sense
- Timing is perfect
- Not a simple process
- Needs full commitment

Why is it needed ?

- Business as usual is not acceptable anymore
- Cost of errors is astronomical
- Y2K and Euro have shown the need
- Best opportunity is now
- Important Y2K and Euro implications

Benefits - Immediate

- Accurate and current information
- Prevent application maintenance errors
- Simplify code changes
- Improve testing procedures
- Simplify application audits

Benefits - Long Term

- Simplify migration to other platforms
- Transform applications by incorporating new technologies
- Simplify planning for large scale application renovation/maintenance

Cost Savings

- Hard Dollar Savings are difficult to quantify
- Soft Dollar Savings
 - Smaller application portfolio
 - Accurate inventory
 - Reduced implementation errors

What is needed ?

- Clean Applications*
- Effective Change Management
- Continuous Monitoring
- Periodic Review*

Start with Clean Applications

- Applications must be clean
 - Otherwise garbage-in-garbage-out
 - Building a house on a shaky foundation
- Remove unneeded components
 - Between 20-40% of code is unused
 - Unused code is useless and expensive
- Manual inventory is impossible

Myths

- A production application is “clean”
 - Can contain missing code
 - Can contain unused programs
 - Mismatched programs (latent bombs)
- Y2K or Euro remediated applications are “clean”
 - See above
 - May not be the actual application. May contain incorrect pointers to components.

Myths (continued)

- Application has been tested, therefore it is “clean”
 - Impossible to test completely
 - some components may only be used infrequently
 - Mismatched components can give you good results
 - Parallel testing is particularly “misleading”

Myths (continued)

- Change management will save us
 - Change management cannot ensure that application is clean to begin with.
 - Dependencies have to be manually determined.

Real Life Examples

- 30% of 7.5 million lines of code at a major utility company in California were shown to be unused
- At a major bank, it was discovered that subroutines were being executed from a library that was no longer in production.
- At an electric company Y2K programs were calling pre-Y2K subroutines.

Real Life examples (cont..)

- At a government agency (in a test of one production application made up of 20 jobs and 96 programs)
 - 3 programs of an application in production were shown to be missing source code
 - 5 programs used out-of-date copybooks
 - 2 programs executed old subroutines

Batting 1.000

- At five data centers where analysis was performed, problems were detected at every single one.
- So far not a single clean one has been found.
- Even if you are clean, how would you know and how would you prove it ?

Component Relationships

- Important to understand different types of relationships and their implications with regard to clean applications.
- Assumptions can be dangerous
- Naming conventions can lull you into a false sense of security

Discussion Focus

- Cleaning up applications
 - Get new and current applications into a clean status
 - Document application relationships
- Periodic review
 - Verify that applications are clean
- Simple Change Management cannot ensure that applications stay “clean”

Clean Application Characteristics

- The starting points (drivers) are known.
- All the component types are known.
- All individual components identified.
- All key info on components available
- Only “permitted” compiler options used
- All inter-relationships between components (explicit, implicit and assumed) are known and tracked .

Clean Application Characteristics (continued)

- No extraneous components present
- Information audited.
- Information certified to be correct.

Creating Clean Applications

- Identify applications to be retained.
- Identify all application drivers.
- Choose the right Software tool.
- Task cannot be done manually.

Doing the Cleanup

- Remove unused components. Be “ruthless” about it.
- Recompile programs if necessary to get them “in-sync”
- Recreate missing source components.
- Rerun software tool “again” to verify.
- Make sure all components have “correct” date-time stamps.

Software Tool Requirements

- Tool should simulate execution paths of applications.
- Tool should track interrelationships.
- Tool should be able handle explicit, implicit and assumed relationships.
- Tool should provide detail and summary documentation.

Keep it Clean

- Use a “Change Management” product to track all changes.
- Periodically run the cleanup tool to verify “clean” status of applications.

Summary

- Clean Management is vital.
- The right way is to start with clean applications.
- Periodically verify an applications “clean” status.
- Know which applications you want to keep.
- You need the right tool.
- Simple “change management” is not sufficient.

Achieving Customer Satisfaction through Requirements Understanding

John Elliott and Peter Raynor-Smith

(c) Copyright 1999 by Defence Evaluation and Research Agency, UK. All Rights Reserved.

Abstract

Customer satisfaction is one of the key drivers in systems development. However, widespread customer satisfaction is not attained largely due to problems of inadequate 'requirements understanding'. This paper describes a process improvement theme and case study that has been directed towards better customer satisfaction through improved through-life requirements engineering and management.

1. Introduction

One key goal of all businesses is to achieve a continuous and high level of customer satisfaction in the delivery of services and/or products. Such satisfaction is believed to be the basis of long term profitability and business growth. In the sphere of computer based system products, customer satisfaction is dependent on how system development projects evolve to build operational product systems that satisfy the perceived and actual customer need and associated system requirements.

Ultimately, successful customer satisfaction depends upon the depth of 'through-life' understanding about the business need and associated user requirements for a future system, and the ability to communicate those requirements to the system developer. In addition, customer satisfaction and confidence depends upon the level of system assurance offered throughout the system development lifecycle. Requirements understanding problems inevitably lead to poor customer-supplier relationships, unnecessary re-works, and overruns in cost and/or time.

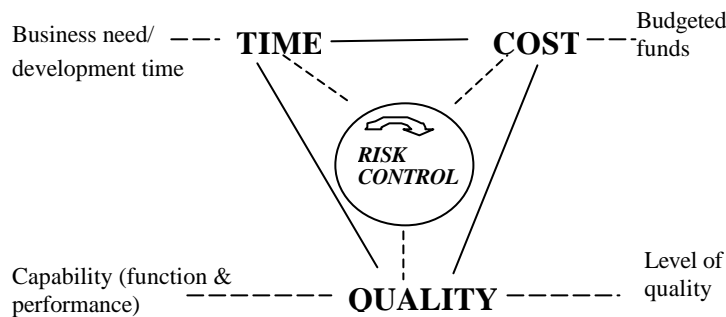
This paper discusses the concepts underpinning customer satisfaction and requirements understanding relevant to software-based system development. In addition, the design of customer-oriented development processes is described together with a process improvement case study and associated experiment. The process improvement experiment was EU project number 23893, REJOICE, whose Final Report [17] can be found at the ESSI VASIE website [18]. The REJOICE experiments and their results have been summarised in Section 5 of this paper.

2. Customer Satisfaction and Requirements Understanding

Customer satisfaction is dependent upon many factors that are associated with the business need, the development project and resultant system product quality. Ultimately the customer is looking for added value to benefit the business operations within a defined timeframe but at an affordable price; hence the customer priority is for an overall *successful business*. The system supplier perspective is to deliver a system within the agreed cost plans to satisfy the customer requirements, thus contributing to the supplier's profit and reputation; hence the supplier priority is for a *successful project*. These different perspectives are typically controlled through inflexible and formal contract management arrangements in the pursuit of a successful project for both customer and supplier. The cornerstone to such 'success' involves an appropriately rigorous approach to 'quality' by customers and suppliers.

Quality may be loosely inferred to mean 'satisfying requirements' embracing the provision of added capability (i.e. improved business function and performance) and any associated trustworthiness or integrity (i.e. continuously performs as intended without harmful 'side-effects' on business services). One key aspect of the quality perspective concerns the customer and supplier agreeing upon a required level of quality to be achieved within defined and understood cost and time constraints. In addition, the quality level must be defined and be subject to some agreed measurement to monitor attainment. Figure 1 highlights the various project viewpoints affecting quality.

Figure 1- Balancing the development achievements



The remaining development project consideration is the level of risk and uncertainty associated with the attainment of the required and agreed quality level; the risk perspective depends upon the available knowledge about the project constraints and their implications. Hence both customer and supplier need to understand the level of risk each is taking within their quality level agreement. In practice, the notion of risk sharing between customers and suppliers is a difficult area that influences the nature of any supporting legally binding contractual arrangements. In summary, both customers and suppliers need to plan and implement compatible quality and risk strategies for the development project. These strategies will need to be reflected in any contractual agreements.

Returning to quality within the customer satisfaction arena, customers need to be assured that defined and measurable final product quality attributes demonstrate that their defined needs and associated requirements are satisfied. Achieving defined product quality depends upon 'getting the system requirements right' and then 'building the product right' to meet these requirements. This is not easy to achieve especially within traditional contracting processes that tend to encourage the communication of requirements through formal documents and review activities. This inflexible and formal approach to agreement and communication is often the main reason why customer and supplier teams fail to be effective in achieving continuous levels of understanding, which is sometimes coloured by a culture of disrespect and mistrust.

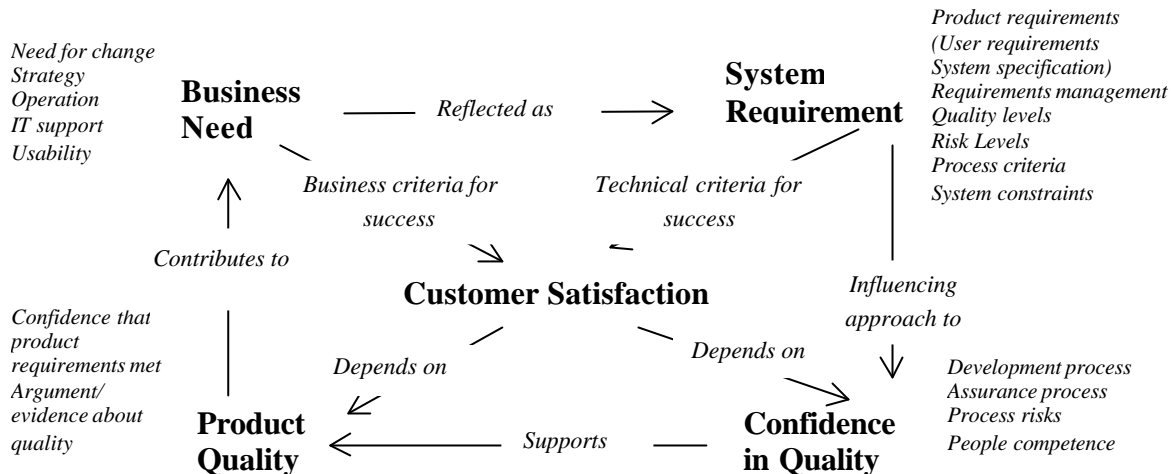
The necessary criteria for customer satisfaction are provided below to further demonstrate the relationship with requirements understanding. Such criteria provide the basis for defining measurement schemes from which to systematically argue and justify whether customer needs and requirements have been adequately satisfied.

- The business need for supporting necessary or desirable (process and information) change must be clearly defined.
- The system requirements must be clear (and error-free) and related to the business need.
- The supplier's development process (for all management, engineering and quality activities) must be consistent with best practice.

- The supplier processes must closely interface with the customer's processes in executing the acquisition and system creation activities.
- The competence and performance of the supplier teams must be of a high standard.
- There must be high visibility of the executing development processes and of the product evolution.
- There must be an ability to change the product development as the requirements are better understood and refined (or even changed due to business reasons).
- The final system product must be compliant with the agreed and understood requirements.
- The final system product must meet defined business needs and added value to the customer's business operations.
- The final system product must have high levels of usability and be easily integrated into customer processes.
- There must be sufficient demonstration regarding the satisfaction of business needs, system requirements and product quality (i.e. overall fitness for purpose).
- The agreed project schedule must be met ensuring that the final system delivery and in-service dates are achieved.
- The project costs must not be changed without full agreement and justification in customer terms.

Partly derived from the criteria above, the customer satisfaction problem domain has four key dimensions, see Figure 2: business need, system requirements, product quality and confidence in quality. This is the basis of a customer satisfaction model.

Figure 2 – Four Domains of Customer Satisfaction



This customer satisfaction model above implies that customer satisfaction is analogous to overall project success. Garrity [1] found that development project success is more than customer satisfaction as success largely reflects a further two considerations of usability and adaptability. *Usability* concerns the wider process considerations beyond system delivery and acceptance embracing the operational experience; this involves different perspectives when applying the new system for individual task support and business organisation performance enhancement. *Adaptability* has a cycle (of planning, doing, filtering and learning) to adjust development progress and direction based on business and development feedback and interaction.

Both success notions of usability and adaptability are vital to achieve longer-term customer satisfaction. Usability, through process analysis, should be a part of the analysis of business need. The design and implementation of customer-oriented processes should be based on an adaptive system model, similar to the Viable Systems Model [2], to represent a flexible and dynamic development approach based on different levels and orders of system-environment learning, feedback and adjustment.

Those parts of the customer satisfaction model that address business need and requirements understanding [13], must ensure that all aspects of user and system requirements are considered. For example, the overall system requirement needs to include the system product requirements as

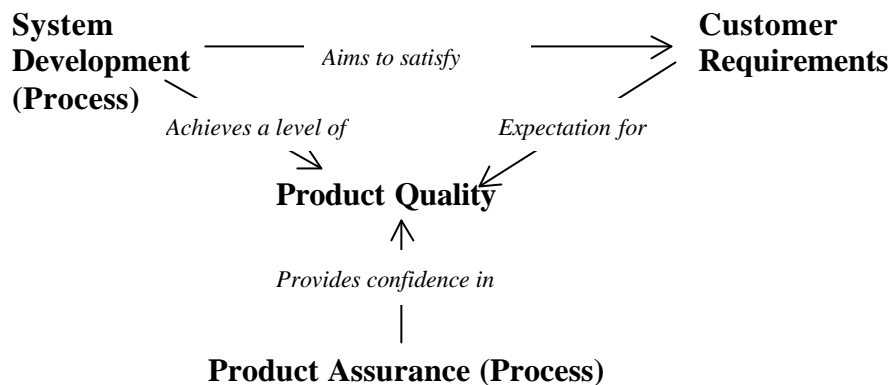
well as those requirements addressing quality and risk levels, development process criteria and the project constraints, e.g. interoperability with existing systems, timescales and costs.

Of key importance to customer satisfaction is the central product quality concept, which loosely means ‘satisfying the customer requirements or need’ throughout the product life cycle, from ‘birth to death’. The product quality requirements will describe a range of external and internal system product attributes; external attributes include its functionality and performance (e.g. speed, reliability, maintainability, safety, security, etc) whereas internal attributes include its architectural structure, portability etc. Different authors such as Fenton and Gillies [3, 4] describe and review different quality models including that developed for the ISO 9126 standard [5].

The key achievement of actual product quality can only be measured by reference to a quality profile [6] that is a weighted representation of each system product attribute. The attribute weights are derived through customer analyses at the beginning and throughout the development project. The satisfaction of product quality is judged by the combined final weighted attributes achieved against that required through prudent use of project resources to address attributes within designs, trade-offs reviews and their validation.

The product quality achievements depend on the required quality and risk target levels, and the design and execution of development (i.e. creating) and assurance (i.e. checking) processes. The relationship between product quality, customer requirements, and system development and assurance processes are emphasised in Figure 3.

Figure 3 - Impact of Product Quality



Assurance involves checking all levels of the design and provides the argument and supporting evidence, (i.e. as system measurements of 'fit for purpose') that the need and requirements have been addressed to the required quality and risk levels. All processes need to follow consensus best practice that has been suitably tailored to the specific development project needs, while taking into account all associated quality and risk levels. These levels are related to the appropriate process and product criteria. The process criteria reflect the degree of development and assurance rigor to be adopted. The product criteria reflect the design criteria to be adopted in system architectures and detailed design. The customer's confidence in the final system product is affected by the visible degree of thoroughness by which the defined and planned processes were followed and executed; this confidence is also affected by the competence and performance of the development (and customer) teams.

3. Customer-oriented Lifecycle Processes Design Attributes

The aim has been to define a technical strategy based upon the fundamental understanding embodied in the customer satisfaction model. The strategy enhances the level of customer satisfaction through improved customer-developer process design with an emphasis on requirements and their understanding. There are three questions to be considered in forming an appropriate technical strategy and in designing a customer-oriented process:

- What are the *attributes* of a customer-oriented lifecycle process?
- How does the customer-oriented process fit relate to current *lifecycle* models?
- What *techniques* are appropriate to be used within a customer-oriented lifecycle process?

Customer-oriented lifecycle process attributes. Based on the concepts in Section 2, the following are the key requirements on which to design a new approach to customer satisfaction. The required attributes are below.

- *Through-life* treatment of system requirements and business need; this will focus attention on the ultimate project goals and success criteria
- Need to be *flexible* to changing customer needs and perspectives; this will encourage effective contracting and working arrangements to be in place that are based on the premise that such change is inevitable and technical agreements will need to change.
- Must be *fast* to react to changing customer perspectives about system requirements; this assists customers to quickly see the impact of their desired changes.
- Enable executable system prototypes to be *visible* and allowing user 'play back'; this enables the customer team to see the evolving product in concrete terms and respond accordingly.
- Need to be able to *roll* the current system solution both forwards and backwards; this assists the speed at which changes (using new or old perspectives) can be played back.
- Need to embrace the whole system *evolution* lifecycle; this will ensure that systems are not viewed as totally new but rather as add-ons or modifications to existing, albeit larger, systems.
- Need to ensure that customers get operational systems as a series of *increments* to meet shorter-term priority needs; this will enable customers to get useful employable systems as a series of incremental deliveries formed within an well-founded overarching business system architecture.
- Need for customer-supplier teams to work in *partnership*; this will enable both parties with separate overall business aims to share a more focused and explicit common project goal within a trusted contractual and working relationship that involves more risk and information sharing, and joint decision making.
- Need effective *communication* between customer and supplier teams; this enables a common and shared understanding about the business need, system requirements, and the development processes and products.
- Need customers and suppliers to be regularly *interactive* about key business and development changes affecting the partnership; this enables an on-going approach to holism, learning and adaptability throughout system evolution.
- Need frequent customer *feedback* to design concepts and system increments prior to final acceptance and in-service use; this will ensure that customers declare timely change based on business use perspectives.
- Need to manage the customer needs and requirements and their *satisfaction* through a flexible yet controllable approach to system planning and its execution; this will focus both parties on the theme of customer satisfaction and project success by on-going requirements understanding.
- Need to provide effective risk and quality control mechanisms to decide about system *fitness*; this will enable customers and suppliers to understand their shared risk and views about fitness prior to in-service-use.

Customer-oriented processes and current lifecycle models. There is much written about development lifecycle strategies, for example, see Somerville, McConnell and Pressman [7, 8, 9]. The main lifecycle variant labels are: Waterfall; V-model; Spiral; Evolutionary prototyping; Incremental/staged delivery; Design to schedule; Design to tools; Commercial of the shelf; and Evolutionary delivery. These variants differ in their attempt at imposing different engineering structures for project management purposes based on implicit premises about flexibility and degree of change, speed of delivery, reuse and integration, and system delivery strategies. The overall conclusion is that these lifecycle variants only partially address the above requirements

for a customer-oriented lifecycle process and a new approach is required to fully encompass customer orientation. The main lifecycles tend to be sequential, static and prescriptive in nature, and assume all projects need the same process structure. No lifecycle adequately represents the real-world dynamic activities between customer and developer, partly a result of their variability and complexity.

Customer-oriented lifecycle techniques. The major techniques need to support the goals for customer satisfaction and in particular requirements understanding. These techniques cover the following process areas: Business analysis; Communication and interaction; Requirements management and engineering; Project and risk management; Quality assurance; Rapid development; Process assessment, e.g. SPICE and CMM; Project and software measurement.

The aim is to populate a customer-oriented lifecycle with a set of relevant techniques, selected from a 'customer-oriented toolkit'. All techniques need to help facilitate the achievement of customer satisfaction and requirements understanding.

4. Proposed Customer-oriented Lifecycle Processes

The customer-oriented lifecycle processes have been based on a technical strategy that, in turn, has been founded on the customer satisfaction understandings in Section 2.

Customer-oriented technical strategy. The proposed strategy is to:

- Define a customer-oriented lifecycle process with the above attributes; that will place an emphasis on well-founded through-life 'requirement understanding' processes.
- Integrate the proposed lifecycle processes into established project, risk and quality management practices; this will involve identifying the tailoring issues surrounding the introduction of a customer-oriented approach into established software practices and local cultures.
- Propose a set of techniques to support the new lifecycle that is appropriate to a project situation.
- Define a means of measuring the effectiveness of the new lifecycle and supporting techniques in business and project terms; this will focus on the cost-effectiveness using criteria about *identifying need*, *communication/interaction* and *requirements control*.
- Ensure that the new customer-oriented approach is focusing on business benefits and be widely applicable; this directs the approach to be geared towards the non-software specialists, needing no specialist tools, knowledge or equipment.

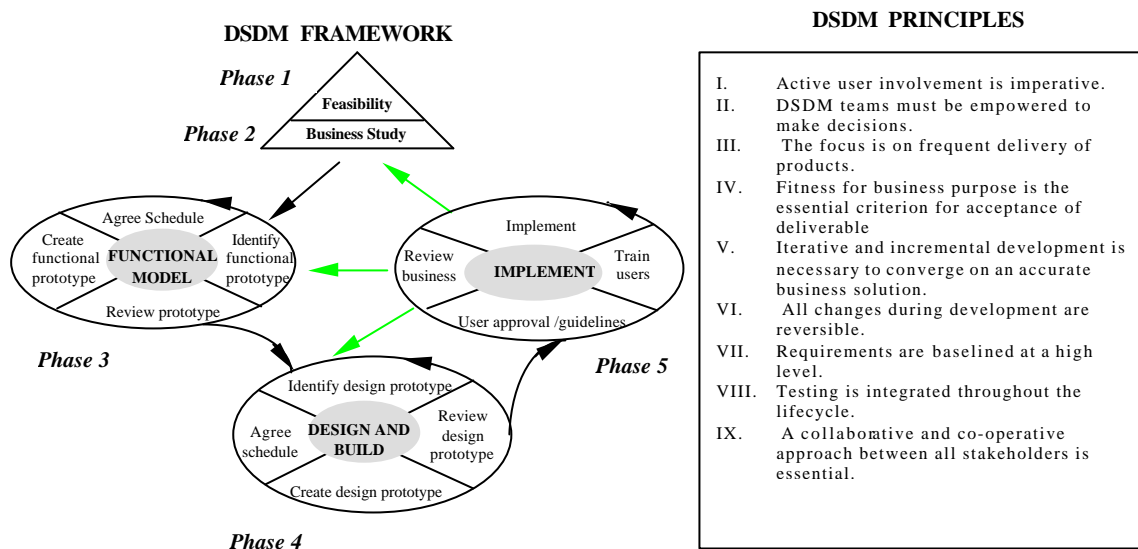
Customer-oriented process overview. The aim is to establish an improved process and set of techniques that will assist customer and supplier to gain a better understanding of initial and changing requirements so that systems are delivered on time, to cost and actually meeting the customer's real need. These techniques will also need to address accomplishing and preserving product quality throughout the product life cycle. The approach combines and utilises techniques from separate strands:

- A customer-oriented lifecycle process supported by fundamental system models that describe requirements understanding concepts and system 'fitness' measurement.
- Use of business analysis techniques such as those exploited in Business Process Re-engineering (BPR) [10] to guide the way in which the customer's real needs are articulated and understood.

- Interactive and iterative approaches such as JAD (Joint Application Development) [11] and RAD (Rapid Application Development) [8, 12] to assist communication and exploration.
- Formalised approaches to capture the statement of requirements, support their management and allow traceability, etc.

The customer-oriented lifecycle process has been based on an adaptation [14] of the Dynamic Systems Development Method (DSDM) [15, 16] framework. DSDM offers a generic lifecycle framework that is geared to being more flexible, faster reacting and dynamic practices involving joint customer-developer working. Figure 4 shows the five DSDM-based customer-oriented lifecycle process phases. The proposed process adaptations to DSDM, as used within the REJOICE process improvement case study, are described further in section 5; these combine and refine Phases 1 and 2 activities.

Figure 4 - DSDM Based Customer-Oriented Lifecycle Process Framework and Principles



- *Phase 1 - Feasibility Study*; An assessment is made as to whether or not the DSDM approach is correct for the anticipated project. [This is not a conventional form of feasibility, i.e. whether the system concept is achievable.]
- *Phase 2 - Business Study*; Provides the foundations on which all subsequent work is based and provides an understanding of the business and technical constraints. [This study is intended to be relatively short with the aim to describe a 'first-cut' high level requirement.]
- *Phase 3 - Functional model*; this activity is broadly equivalent to a functional specification, but expressed using an executable prototype with some documentation support.
- *Phase 4 - Design and build*; this activity is refining the functionality to reflect non-functional and other quality/integrity requirements; the detailed designs are as executable prototypes but with improving quality attributes, supported by essential documentation.
- *Phase 5 - Implement*: this activity applying the product within a series of systems trials ultimately being accepted in the operational environment.

The essence of this approach is for the customer and developer to work in partnership ensuring that the needs and requirements are well understood by all. The system is allowed evolve in terms of refining prototypes resulting in useable increments. The strategy is to be flexible and adaptive to changing requirements and to progressively build quality into the evolving product. The customer-development interactions occur throughout allowing for learning, feedback and adapting to influence development directions. The risk of the flexibility offered needs to be countered through the application of sufficient management and quality assurance practices incorporating process and product checks with sufficient traceable documentation. This approach is to some extent dependent on effective tool-sets in order to gain the customer satisfaction benefits.

5. Process Improvement Case Study

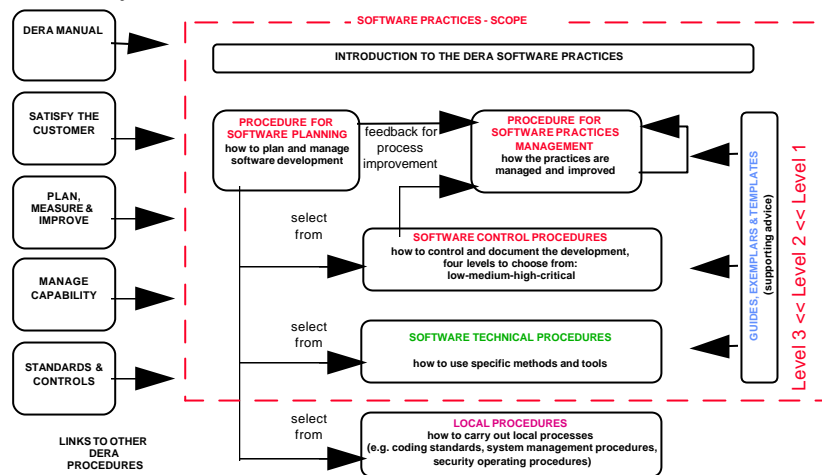
A case study to examine the effectiveness of the new proposed approach to customer satisfaction and requirements understanding was undertaken as an EU funded process improvement experiment (PIE), referred to as REJOICE, ESSI Project 23893. The purpose of the PIE was to demonstrate whether the new customer-oriented process could provide the business benefits sought as improvement goals.

There are various elements to the experiment:

- Business context.
- Improvement goals.
- Proposed process.
- Experimental considerations.
- Results and assessment.

Business context. The experiment was set in the UK Defence Evaluation and Research Agency's (DERA's) System and Software Engineering Centre (SEC). The SEC is an autonomous development and consultancy business that largely serves the defence system businesses within DERA and the UK Ministry of Defence. The SEC is associated with a very wide range of systems for high technology research, system requirements and design modelling, tool development and operational activities. The SEC operates within a highly controlled business management culture (based on the ISO 9000 series) and its activities are regularly subjected to process assessments (e.g. ISO, CMM, SPICE, EFQM-BEM). The SEC has a 'maturing' software culture supported by its DERA Software Practices, as highlighted in Figure 5. The DERA practices incorporate an in-built measurement system.

Figure 5 - DERA SEC Software Practice Structure



Improvement goals. The SEC is striving to achieve the highest levels of CMM maturity (currently achieving Level 3 in some areas) for all its widespread activities supported by the use of SPICE to develop excellence in particular project domains. There were a number of improvement areas identified from various process assessments. This included those concerned with customer relations and ensuring that the SEC met customer needs and requirements. The relevant ‘customer-related’ goals to be satisfied through an improved approach to requirements understanding were:

- 20% more customer satisfaction.
- No extra effort on requirements activities.
- 15% decrease in requirements generated problem (i.e. less reworks).

Proposed process. The customer-oriented lifecycle process, an adaptation of DSDM as shown in Figure4, was applied within specific development projects. The adaptation was to combine Phases 1 and 2 of DSDM into a single phase, 'User Requirements Study'. The reason was to remove the DSDM suitability analysis (less important to the REJOICE goals than to rapid application development objectives) and to increase the focus on the feasibility and definition of user requirements against a real, and rigorously studied, strategic need for business change. Hence, this new phase focuses on the communication, understanding, elicitation and high level capture of business needs and requirements. In addition, before the adapted DSDM lifecycle process (referred to as the *REJOICE process*) can be applied, further DSDM 'tailoring' considerations need to be addressed:

- How can the flexible proposed process be utilised within a high-control business and quality management culture?
- What standardisation process details should be defined and to what level of detail?
- How do you define the exact process incorporating methods and tools to apply to a specific project?

It should be stressed that the new customer-oriented process represents a major shift in development culture, a major issue for the REJOICE experiment. In support of the new process, a set of specific methods and tools were selected from which the experiment process details were selected. There was an emphasis on business analysis (e.g. BPR), interaction management and

facilitation (e.g. JAD), design methodology (e.g. object-orientation) and requirements management support (e.g. procedures and tools).

Experimental considerations. The experimentation was divided into four parts:

- Experiment 1 - Defining, tailoring and introducing the new customer-oriented ‘REJOICE’ process.
- Experiment 2 – Partial Application of the REJOICE process to the development of a Requirements Modelling Tool.
- Experiment 3 - Applying and measuring the impact of the ‘REJOICE’ process during the development of a DERA Intranet based CMM Self-Assessment Tool.
- Experiment 4 - Comparing the ‘REJOICE’ process with the existing development process during the development of a DERA Intranet based CMM Self-Assessment Tool.

Each experiment had its own design that included a number of specific hypotheses to be tested and an associated measurement scheme, each of which was linked to the improvement goals. Overall the measurement strategy included maximising the use of qualitative observations backed up by argument based on valuable experience identifying the issues, in addition to collecting quantitative measures. The data collection involved a combination of surveys, interviews, project resource extracts and tracking what processes were being implemented in some measurable detail. The major experimental part was the application of the new process to be applied to two tool development projects. Each project had specific and well-informed customer teams; one project was a requirement modelling tool and the other was a CMM assessment support tool. The outline measurement scheme to examine the new process is shown below (more details are described in [17]).

| Goal | Area/Factor | Metrics: |
|---|--|---|
| <ul style="list-style-type: none"> • 20% increase in satisfying customer needs | <p>Customer Satisfaction: meet need; confidence in product; confidence in process/people</p> <p>Product Effectiveness: product quality claimed; demonstration of quality</p> | <p>Satisfaction (score) with project, product, process, people No. of prototype releases - planned, actual No. of the original satisfied/unsatisfied requirements</p> <p>No of requirements changed No of requirements priority changes No of evolution's of requirements</p> |
| <ul style="list-style-type: none"> • No <u>change</u> in costs of requirements activity | <p>Project efficiency: process definition; process cost; people impact</p> | <p>Time spent in customer interactions Number of customer interactions Time spent demonstrating models/prototypes</p> |
| <ul style="list-style-type: none"> • 15% decrease in problems due to poor requirements understanding | <p>Project efficiency: requirement defects; people interaction; process cost impact</p> | <p>Number of requirements not satisfied Effort spent satisfying incorrect requirements</p> |

Experimental Results. The main results of the four experiments are detailed in the REJOICE Final Report [17] that provides detailed qualitative and quantitative (measurements) evidence presented in a form that argues about the validity of the various customer-oriented process hypotheses. The overall results are now briefly summarised in the following table.

| Experiment Description | Main Results: |
|---|--|
| Experiment 1 Defining, tailoring and introducing the customer-oriented process. | <ul style="list-style-type: none"> • Successive levels of tailoring are involved - they are difficult to clearly define • The DSDM based customer oriented framework is 'loosely' defined and requires further refinement and instantiation to be employable • The new DSDM based process does not fit easily with existing Quality Systems • Detailed DSDM based processes cannot be fully prescribed due to the highly iterative processes involved that is dependent on actual product development progress • Detailed project planning cannot be achieved: plans need to stay at a high level or they will lag behind the actual development |
| Experiment 2 Applying and measuring the impact of the new customer-oriented process: Requirement Tool Project | <ul style="list-style-type: none"> • The pragmatic use of principles leads to a 'fit for purpose' product • 'High level' user requirements are difficult to resolve and manage contractually • The use of prototyping techniques are very effective • The contract requirements would not have been met if traditional processes used |
| Experiment 3 Applying and measuring the impact of the new customer-oriented 'REJOICE' process: CMM Assessment Tool Project | <ul style="list-style-type: none"> • There was good 'buy-in' by the development team • There were high levels of user involvement • There was a high level of user satisfaction with the final product • The users sometimes resented the demands on their time • The team emphasis on development of product means documentation/testing suffers unless control exercised; this may be a problem for longer term customer satisfaction • Any organisation and culture changes are non-trivial • It was difficult to control and plan prototyping • It was difficult to monitor project progress with traditional management techniques • The development team was not used to empowerment and they tended to perceive a lack of direction and management |
| Experiment 4 Comparing the new customer-oriented 'REJOICE' process with the existing traditional development process. | <ul style="list-style-type: none"> • It is difficult to compare results with 'traditional' methods due to non-equivalence with stages in 'waterfall' and variants. • Customer surveys provided evidence of improved satisfaction • The REJOICE process was found to be more efficient than traditional methods in terms of required functionality achieved for developer effort • If the development had followed the existing traditional process, that may have led to the development of an altogether different tool, not taking into account real business need • The longer term customer satisfaction advantages are more difficult to assess • The REJOICE process developed products may be more difficult to maintain and evolve |

The collective evidence from all these experiments provides the basis for deriving the lessons that have been learnt within the REJOICE process improvement case study in terms of the technological and business impact of the new DSDM-based REJOICE process. As in the REJOICE Final Report [17], these lessons are now described in terms of these technological and business viewpoints.

Lessons learnt - technological viewpoint. This viewpoint assesses the impact of the new process in relation to current software practices and their evolution. The lessons are:

- Adoption by the SEC of a new, evolutionary yet controlled lifecycle approach (where appropriate to the projects) is expected to lead to improved customer satisfaction.
- DSDM offers a useful set of concepts (sensible principles, flexible requirements philosophy, strong user and end product focus) that will advance the SEC best practices.
- DSDM is not only suitable for 'RAD type' projects but its concepts can be integrated, in full or in part, into more traditional lifecycle approaches.

- The integration of the DSDM based process within a traditional ISO 9000 quality controlled software development operation is non-trivial, unless DSDM is used to do RAD developments only.
- Commonly available tools generally support the basic DSDM based REJOICE process although more sophisticated model based tools are needed that facilitate effective user-modelling interaction (to study requirements and acceptance testing issues).

Overall, the technological lessons about the DSDM based REJOICE processes are fundamental. More radical software lifecycles are designed to improve customer-developer relations. These require new ways of thinking about project control and tool based cultures. There is clear evidence that the REJOICE process is sufficiently mature and does indeed enhance customer satisfaction, assuming that a joint product-focused management approach is taken by both customers and developers. In short, the REJOICE process offers clear claimed benefits when used in part or in full, but there are a number of non-trivial project and quality management issues to overcome.

Lessons learnt - business viewpoint. This viewpoint assesses the impact of the new process in relation to business goals and activities. The lessons are:

- Customer satisfaction and the attendant advantages are likely to be achieved by the using the DSDM based REJOICE Process.
- The REJOICE process is likely to provide cost saving gains in the efficiency of requirements-based activities, dependant on project complexity and associated implementation issues.
- The REJOICE process requires a co-operative product focused management approach.
- Definition and management of contractual boundaries will be challenging.
- Cultural changes may be difficult to manage.
- Consider applying DSDM techniques to smaller projects until confidence is gained.
- A REJOICE type process will increase business opportunities through improved customer relations.

Overall, many software businesses, often Small Medium Enterprises, should benefit from the DSDM-based REJOICE concepts, process and techniques in terms of customer satisfaction and requirements efficiencies. However, the degree of success will depend upon the organisation and customer culture, the appropriate application to suitably complex projects and an effective use of available software technologies. In short, the REJOICE process framework is well founded but its success critically depends on the management of people and technical resources during any development project implementation.

6. Summary

This paper has described the underpinnings and development of a customer and requirements focused 'REJOICE' process that has been adopted from DSDM. The underpinning arises from the evolving development of an innovative customer satisfaction and requirements understanding model that has a key system measurement component. A new customer oriented lifecycle process has been defined and examined within an EU funded process improvement experiment, REJOICE. REJOICE has focused on the business impact of a requirements-oriented process improvement geared to improve customer satisfaction; the business goals include improved customer satisfaction and cost effective requirements management. The experimental findings support the main hypothesis that the flexible process should yield the business benefits

suggested; however a careful approach to process introduction is required as a new cultural approach to customer-supplier partnerships is critical. If implemented well, both customers and suppliers should reap major benefits.

7. Acknowledgements and Disclaimers

The authors would like to acknowledge the European Commission for funding the Process Improvement Experiment, REJOICE, Project 23893, supplemented by internal Defence Evaluation and Research Agency funding on the general concepts underpinning requirements understanding.

The views expressed in this paper are entirely those of the authors and do not represent the views, policy or understanding of any other person or official body. Further details can be requested from the Defence Evaluation and Research Agency (DERA Malvern), Systems and Software Engineering Centre, Tel: +44 1684-895161, E-Mail: jjelliott@dera.gov.uk.

8. References

- [1] Garrity, E.J., Saunders, G.L., "Information Systems Success Measurement", IDEA Group, 1998.
- [2] Beer, S., "Decision and Control", John Wiley and Sons, 1966.
- [3] Fenton, N. E., Pfleeger S. L., "Software Metrics", 2nd Ed, Thomson Computer Press, 1997.
- [4] Gillies, A., "Software Quality - Theory and Management", Chapman and Hall, 1992.
- [5] ISO 9126, "Software Product Evaluation", 1992
- [6] Van Ekris, J., "Towards business oriented questionnaires for the specification of software product quality", ESCOM-ENCRESS 1998 Proceedings, May 1998, pp230-238.
- [7] Somerville, I., "Software Engineering", Addison Welsey, 1996.
- [8] McConnell, S., "Rapid Development", Microsoft Press, 1996.
- [9] Pressman, R. S., "Software Engineering - A Practitioner's Approach", McGraw Hill, 1997.
- [10] MacDonald, J., "Understanding Business Process Re-engineering", Hodder & Stoughton, 1995.
- [11] Bell, S., Wood-Harper, T., "Rapid Information Systems Development - System Development in an Imperfect World", Second Ed., McGraw-Hill, 1998.
- [12] Martin J., "Rapid Application Development", New York: Macmillian, 1991.
- [13] Elliott, J. J., "System Understanding Reference Model", DERA Report, 1999.
- [14] Raynor-Smith, P. M., "REJOICE Process", DERA Report, 1998.
- [15] DSDM Consortium , "DSDM Manual", 1996.
- [16] Stapleton J., "Dynamic Systems Development Method", Addison-Wesley, 1997.
- [17] Raynor-Smith, P. M., Elliott J. J., REJOICE Final Report, Version 1.0, May 1999
- [18] ESSI VASIE website: [http:// www.cordis.lu/esprit/src/stessi.htm](http://www.cordis.lu/esprit/src/stessi.htm)

QWE' 99

QWE'99 Presentation, Nov' 99

Achieving Customer Satisfaction through Requirements Understanding

John Elliott and Peter Raynor-Smith,

Systems and Software Engineering Centre
Defence Evaluation & Research Agency (DERA), UK

For further info - Email: jjelliott@dera.gov.uk, pmrsmith@dera.gov.uk

SEC
DERA Systems & Software Engineering Centre

DERA V1.1

Structure

- Introduction
- Concepts
- Customer-oriented Process
- Experiment Design
- Experiment Conclusions
- Lessons and Progress
- Summaries

SEC
DERA Systems & Software Engineering Centre

DERA V1.2

Nov 1999

Introduction-1

- Process Improvement Study consisting of:
 - REJOICE - EC PIE (Process Improvement Experiment)
 - Internal 'requirements understanding' research
- Objective: To measure the effectiveness of a new *customer and requirements-oriented approach* designed to provide key business improvements
- Process improvement driven by business needs
- Completed May 1999

Introduction-2

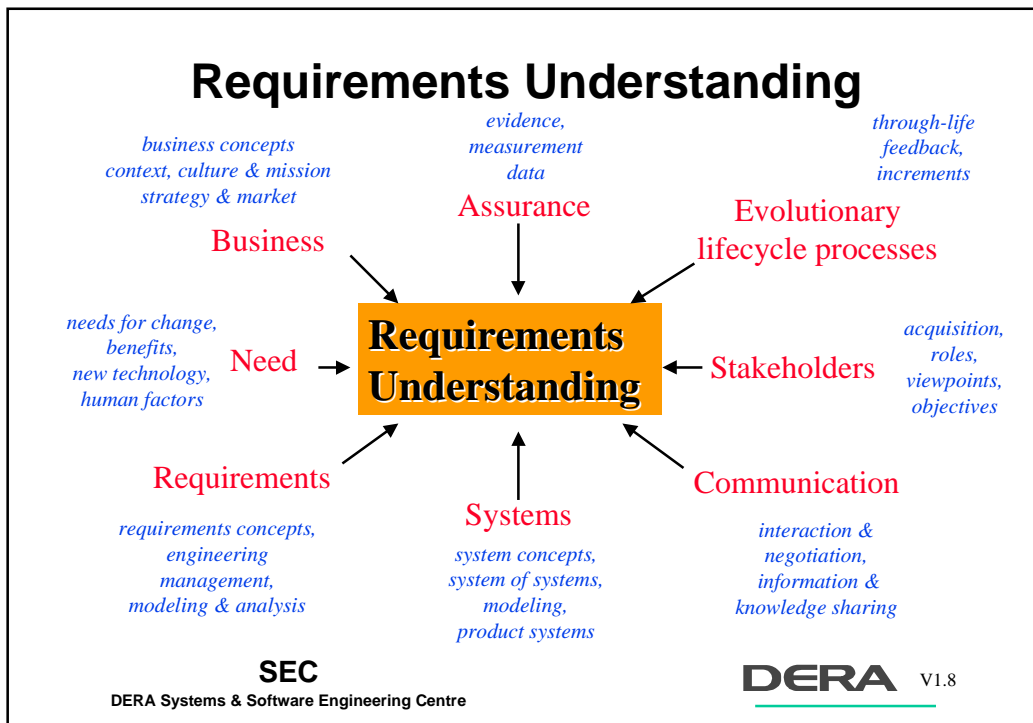
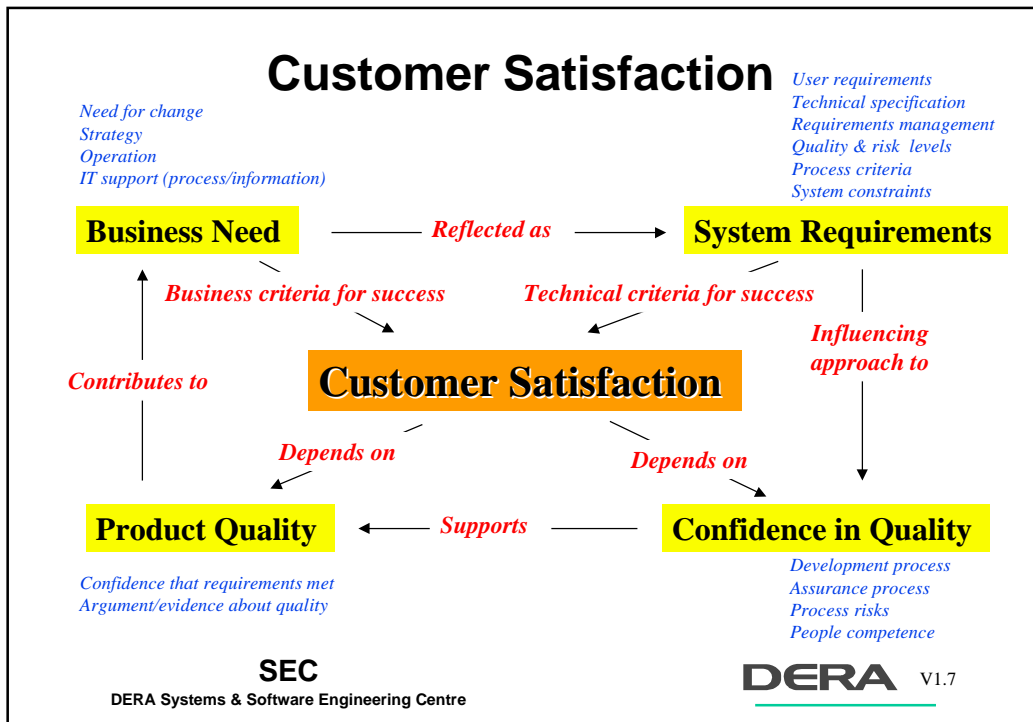
- The Systems and Software Engineering Centre (SEC) is an *autonomous* system and software development and consultancy business division
- *Diverse* software applications
 - technological research, system requirements and design modeling, and operational activities
- Operates within a *highly controlled business management culture* (based on ISO 9000)
- Regularly assessed '*maturing*' software practices and culture, e.g. ISO, CMM, SPICE, EFQM-BEM

Introduction-3

- **Process improvement strategy** includes:
 - customer management
 - requirements management
- The '**customer-related' goals** include:
 - improved customer *satisfaction*
 - no extra *effort* on requirements activities
 - less requirements generated *problems*

Concepts

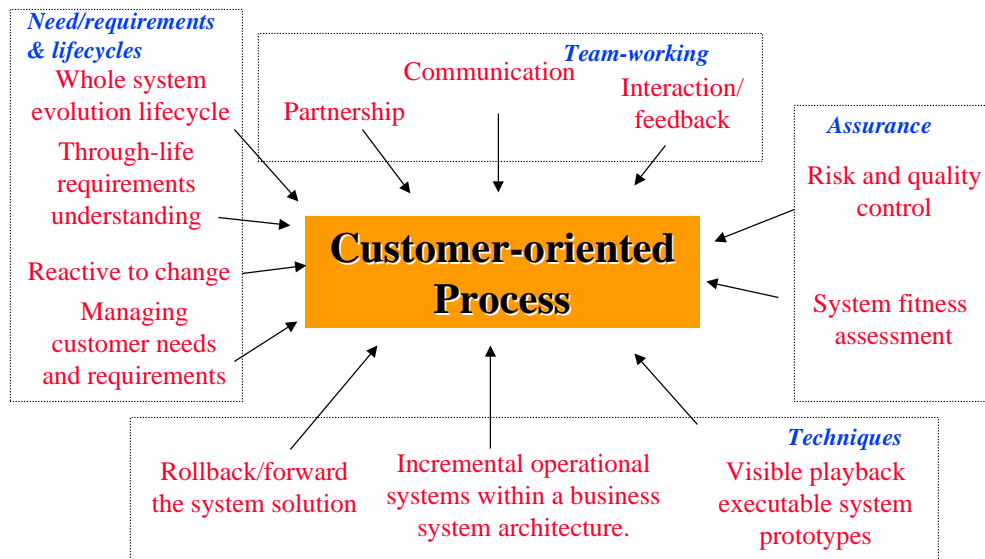
- **Customer satisfaction** is achieved by providing *usable* and *added-value* services and products to meet the *full expectations* of customers to provide known and predicted *benefits*
- **Requirements understanding** is ensuring that all parties linked to customers and suppliers (e.g. all stakeholders) *establish, define, maintain* and *deliver* the same *valid understanding* about *customer need* throughout the *life-time* of the service or product
- Both are closely related to the notion of *quality*



Customer-oriented Process (CoP)

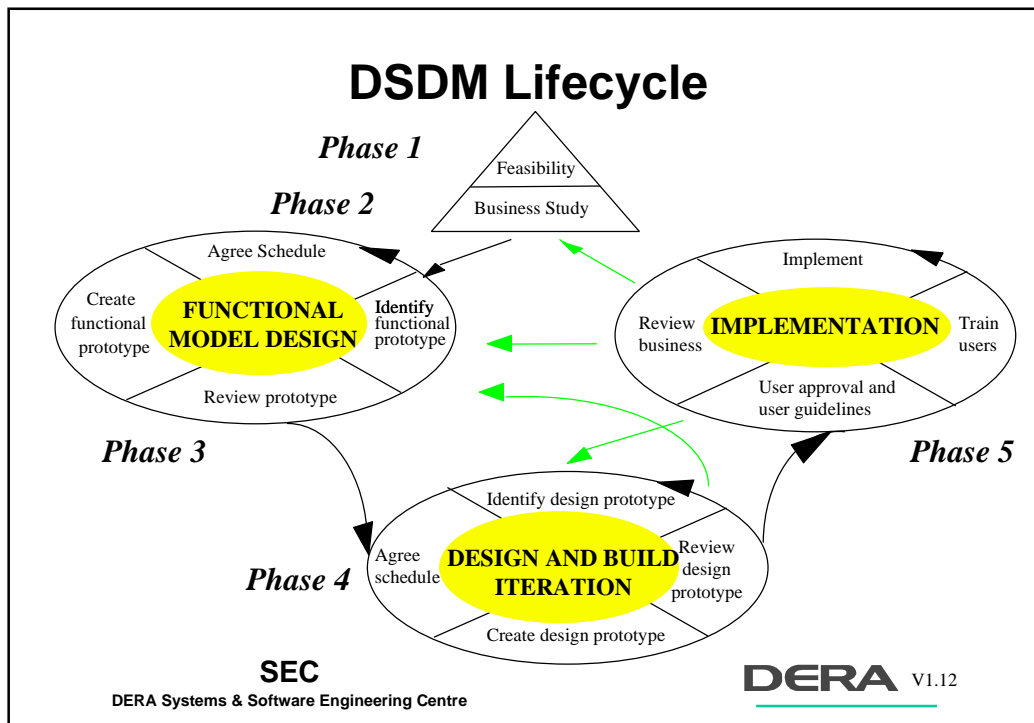
- The customer-oriented process achieved by :
 - better *requirements understanding*
 - more efficient and effective *requirements management*
 - wide *applicability* not requiring specialised skills
- Need to consider:
 - *process criteria* as CoP requirements
 - appropriate *lifecycle* and *techniques*

Customer-oriented Process Criteria



Customer-oriented Process Lifecycle

- **Dynamic Systems Development Method** (DSDM) framework offers a generic lifecycle geared to *flexible* and *dynamic practices* involving *joint customer-developer working*
 - common lifecycle variants tend to be inadequate - sequential, static and prescriptive
- DSDM is a customer-oriented process
 - *through-life* 'requirement understanding'
 - supports *need identification, communication* and *requirements control*
 - concepts are *widely applicable*



DSDM Phases

- *Phase 1 - Feasibility Study* - use DSDM?
- *Phase 2 - Business Study* - understand business and technical constraints to determine high level user requirements
- *Phase 3 - Functional model* - develop functional specification using executable prototypes
- *Phase 4 - Design and build* - refine the functional model to reflect non-functional requirements
- *Phase 5 - Implement* - install and use system(s) with user training

SEC

DERA Systems & Software Engineering Centre

DERA V1.13

DSDM Principles

- User involvement
- Team empowerment
- Frequent delivery
- Fitness for business purpose
- Iterative and incremental development
- All changes reversible
- Requirements are at a high level
- Testing is through-life
- Collaborative and co-operative approach

SEC

DERA Systems & Software Engineering Centre

DERA V1.14

Experiment Design-1

- To assess the **impact** of a new DSDM-based REJOICE process [*small adaptation*] on meeting customer-oriented business goals
- **Experimental framework** developed to define and test/measure hypotheses about:
 - **introducing** DSDM
 - **partial use** of DSDM (e.g. principles)
 - **full use** of DSDM (e.g. process)
 - **comparisons** with traditional lifecycle approaches

Experiment Design-2

- Measurement of the **process impact** on business goals
 - **customer satisfaction** and **product effectiveness**
 - **project/process efficiency** about requirements
- Mixed measures:
 - experience/argument, quantitative, qualitative
- Data
 - survey/interview, project resources, process tracking

| Goal | Area/Factor | Metrics: |
|---|---|---|
| <ul style="list-style-type: none"> Increase in satisfying customer needs | <p><i>Customer Satisfaction:</i> meet need; confidence in product; confidence in process/people</p> | <p>Satisfaction (score) with project, product, process, people No. of prototype releases - planned, actual No. of the original satisfied/unsatisfied requirements</p> |
| | <p><i>Product Effectiveness:</i> product quality claimed; demonstration of quality</p> | <p>No of requirements changed No of requirements priority changes No of evolution's of requirements</p> |
| <ul style="list-style-type: none"> No change in costs of requirements activity | <p><i>Project efficiency:</i> process definition; process cost; people impact</p> | <p>Time spent in customer interactions Number of customer interactions Time spent demonstrating models/prototypes</p> |
| | <p><i>Project efficiency:</i> requirement defects; people interaction; process cost impact</p> | <p>Number of requirements not satisfied Effort spent satisfying incorrect requirements</p> |
| <ul style="list-style-type: none"> Decrease in problems due to poor requirements understanding | <p><i>Project efficiency:</i> requirement defects; people interaction; process cost impact</p> | <p>Number of requirements not satisfied Effort spent satisfying incorrect requirements</p> |
| <p>SEC DERA Systems & Software Engineering Centre</p> | | <p>DERA V1.17</p> |

| | | |
|--|--|--------------------------|
| <h2>Experiment Conclusions - Business</h2> <ul style="list-style-type: none"> The DSDM-based process is expected to lead to <i>improved cost-effective customer satisfaction</i> The DSDM-based process is likely to provide <i>cost savings in through-life requirement activities</i> depending on project and stakeholder complexities Definition and management of <i>contractual boundaries</i> will be challenging DSDM-based process requires a <i>co-operative product-focus</i> <i>Cultural changes</i> may be difficult to manage | <p>SEC DERA Systems & Software Engineering Centre</p> | <p>DERA V1.18</p> |
|--|--|--------------------------|

Experiment Conclusions - Technical-1

- DSDM offers a *new, evolutionary yet controlled lifecycle approach*
- DSDM offers a *useful set of concepts and techniques* to be used in whole or part to advance traditional best practice *beyond RAD*
 - *Concepts*: good principles, flexible requirements philosophy, strong user and end product focus
 - *Techniques*: facilitation, time-boxing, prototyping

Experiment Conclusions - Technical-2

- *Tailoring DSDM* into a local development project environment presents key difficulties
- *DSDM integration* within a traditional ISO 9000 quality controlled software is non-trivial
- Commonly *available tools* generally support the basic DSDM-based process
 - enhanced *model based tools* are needed that facilitate effective *user-modelling interaction* (to study requirements and acceptance testing issues)

Lessons for SMEs-1

- The DSDM process is so radically different that the *scope (size) of the project* should be kept small until confidence is gained
- Can the organisation tolerate and support the concepts of *empowerment and cultural change*?
- Will the customers and users be *amenable to the process*?

SEC

DERA Systems & Software Engineering Centre

DERA V1.21

Lessons for SMEs-2

- Depending on the culture, keep the development to *involve controllable familiar organisations*, thus avoiding complicated contractual arrangements and inflexible working practices
- Introduce facilitated workshops, time-boxing and more rigorous requirement prioritisation as techniques to *enhance more traditional lifecycles*
- *Iterative and incremental processes* are difficult to manage and monitor

SEC

DERA Systems & Software Engineering Centre

DERA V1.22

Progress towards Business Goals

- Customer satisfaction improved
 - *collaborative product focus* fundamental
- Through-life requirements efficiency likely
 - requirements largely *represented by models and prototypes* not documentation
 - leads to *same effort* depending on project complexity and associated DSDM implementation approach
 - *flexible lower cost approach to rework* through closer joint working and product monitoring

Business Summary - Final Report....

“Overall, many software businesses, often Small Medium Enterprises, should *benefit from the DSDM-based concepts, process and techniques* in terms of customer satisfaction and requirements efficiencies. However, the degree of success will depend upon the organisation and *customer culture*, the appropriate application to *suitably complex projects* and an effective use of available *software technologies*. In short, the DSDM process framework is well founded but its success critically depends on the *management of people and technical resources* during any development project implementation. “

Technical Summary - Final Report....

“Overall, the technological lessons about the DSDM processes are fundamental. More radical software lifecycles are designed to improve customer-developer relations. These require new ways of thinking about *project control and tool based cultures*. There is clear evidence that the DSDM process is sufficiently mature and does indeed enhance customer satisfaction, assuming that a *joint product-focused management approach* is taken by both customers and developers. In short, the REJOICE process offers clear claimed benefits when used in part or in full, but there are a number of *non-trivial project and quality management issues* to overcome.”

SEC

DERA Systems & Software Engineering Centre

DERA V1.25

Automated Code Inspection

*A practical approach to improve code inspection
efficiency*

Banco Río de la Plata work experience

Marcelo Dalceggio

mdalceggio@intranet.bancorio.com.ar

Alvaro Ruiz de
Mendarozqueta

aruizdemendarozqueta@intranet.bancorio.com.ar

Banco Río de la Plata S.A.

Grupo Banco Santander Central Hispano

Executive Summary

Banco Rio de la Plata is one of the major private banks of Argentina, with 250 branches executing one million transactions a day and providing universal banking services to local, regional, and international customers.

In 1997 it faced the challenging Y2K problem. It contracted different service providers to assess and remediate the code using automated tools. Some applications were replaced and others were solved in-house by manual process. The code was fixed using the 'windowing' technique (This solution carries a long term risk. 'Windowing' logic changes program code but not data). It performed Y2K baseline (regression) testing and post-Y2K testing. More than 7.000 kloc of Cobol and Assembler code were reviewed and a lot of new code was added to the legacy system making it vulnerable to the introduction of new defects, because those changes made the code larger and more complex. In this context, QA staff had the mission to provide a process that would let the Bank ensure that maintenance activities (changes and enhancements) wouldn't infect the programs. This process was called '*Y2K Contamination Control*'.

It was almost impossible to perform this activity manually. A tool-based inspection process was developed in order to achieve the goal. Good results encouraged QA staff to extend it to address other common defects (not only correct date processing) such as code standards & good coding practices violations.

This paper summarizes this process and how it was enhanced to become part of the ongoing software quality process, increasing productivity and reducing costs.

How the idea originated

Code Inspections advantages are very well known but they are very hard to implement, cost a lot, and critical projects not always have the opportunity to apply them.

During Y2K project we developed a 'Process' in order to control code contamination within the maintenance activities

Both experiences triggered the original idea

The Company

Grupo Santander has been for years the leader financial group in Spain and with the last merger with Banco Central Hispano has become the new *Grupo BSCH*, one of the biggest financial groups in the unified Europe today.

In Latin America, it has the major foreign bank commercial net with banks in many countries of the region. Two years ago, Grupo Santander acquired *Banco Rio de la Plata* in Argentina, one of the major private banks in the country, which with its 250 branches executing one million transactions a day provides universal banking services to local, regional, and international customers.

Since its acquisition, Banco Rio de la Plata has been continuously creating new and sophisticated financial products, demanding legacy systems transformations while increasing productivity and reducing costs to maintain a competitive edge and operational excellence in today's business world. Best software engineering practices are helping the organization to achieve these goals.

Y2K Experience

Y2K Project

The Y2K Conversion Project applied windowing technique in order to solve the two-digit data for the year. This technique reduced costs and allowed an easier implementation but one of its major problems is the risk associated with maintenance. During the modification

activities the modified code with windowing technique could easily be changed undoing the Y2K fix.

With the goal of reducing those risks we developed what we call 'Y2K Contamination Control Process'

Contamination Control

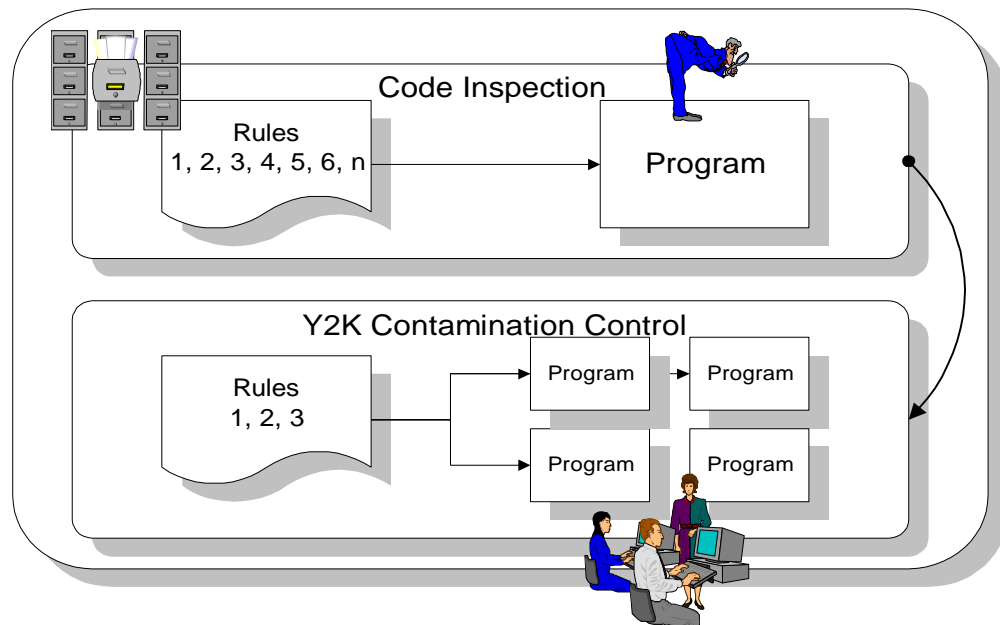
We established code revisions to detect possible violations to the Y2K fix during maintenance activities. We did a survey looking for different ways of introducing a defect within the code or certain situations that might be a risk for future modifications.

These few Y2K compliance rules had to be applied in all the programs, not only at the end of Y2K conversion project, but also during the everyday maintenance work. This was the real challenge and for that volume of work was impossible to apply code inspections in the 'Fagan style'. Manual code inspections were unfeasible.

We developed a software-assisted inspection process with a very simple tool. We parsed the programs with it and in every situation of a probable 'fix violation', a QA staff analyzed the very nature of the problem.

We obtained very satisfactory outcomes and the results presented the following characteristics:

- Every system was reviewed and the results were reported to the managers.
- A weekly revision is being performed for all the new and modified programs.
- The final costs are cheaper than those of the consultant firms running in the market.
- Some findings were checked with manual code inspections and with consultant firms. The results were optimal.
- The contamination control process has to continue as long as windowing fixes remain within the code.
- The following picture shows how we moved from the manual code inspections (many rules in one program) into a software-assisted contamination control (few and stable rules in all the programs).



Some problems with the manual code inspections

Code Inspections advantages are very well known but very hard to implement, cost a lot, and critical projects not always have the opportunity to apply them.

These are the main advantages:

- Defects are detected before the testing process, saving time and money.
- Identifies defects that testing misses.
- Gives complete code coverage.
- Finds root cause defects (no symptoms).
- Additional advantages: Improvement opportunities, Good & Bad practices are finally learned.

Other difficulties with personnel

- Programmers are not used to sharing code, neither during inspections, nor during coding activities.
- Programming as an artistic activity prevails.
- Managers encourage heroic programmers and schedule-driven results.
- Standards are not usually used. Programs are very big and count on no documentation.

Difficulties in Manual Code Inspections

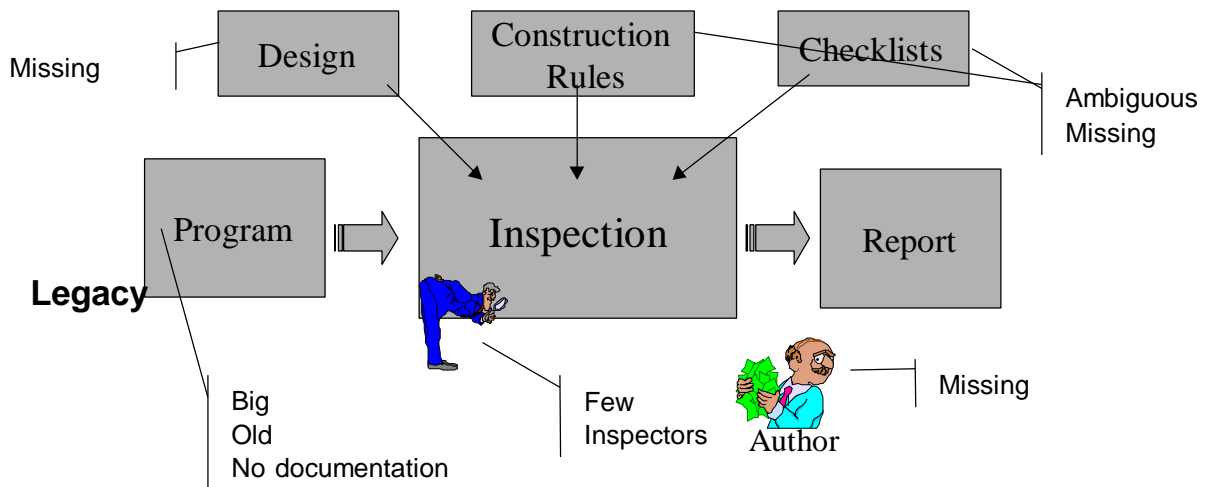
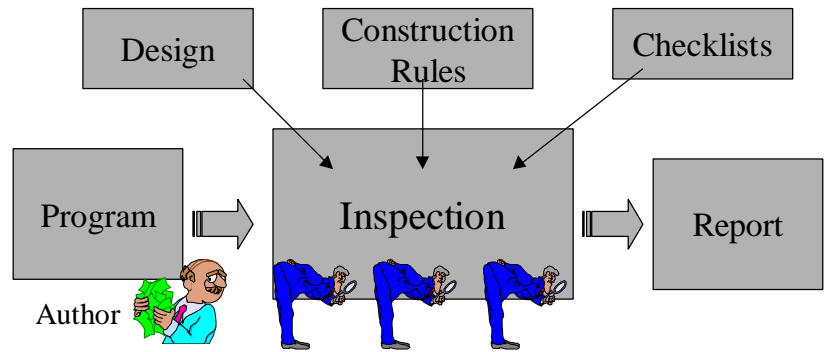
In manual code inspections the well-known authors recommend the following components:

the program that will be the object of the inspection, the design document that originated the program, the

construction rules taken into account during the programming activities, the checklists for the inspection, the author's program and the inspectors.

In our environment we have very old programs. It is very hard to find the design documents. The author may no longer belong to the bank or may be doing other activities rather than programming. (He might not even remember the program structure). It is also very difficult to trace the rules applied during programming. Some of them depend on projects or individual criteria. Checklists include a lot of ambiguity.

Inspectors candidates are assigned to critical projects. They have demanded skills which makes it impossible to assign them to inspections.



Difficulties related to our work environment

Our System Department creates or modifies an average of 900 programs per month. Each program has an average size of 1000 lines of code (LOC).

The monthly demand for inspections is:

$$900 \text{ pgms} * 1\text{KLOC/pgm} = 900.000 \text{ LOC monthly}$$

Taking into account that industry considers an average inspection rate of 100 lines of code reviewed per hour, the total amount of hours needed in our case is:

$$900.000 \text{ LOC} / 100 \text{ LOC/hour} = 9.000 \text{ hours}$$

Assuming that we would like to inspect the total amount of programs we need:

$$9.000 \text{ hours} / 160 \text{ hours/staff} = 56 \text{ staff}$$

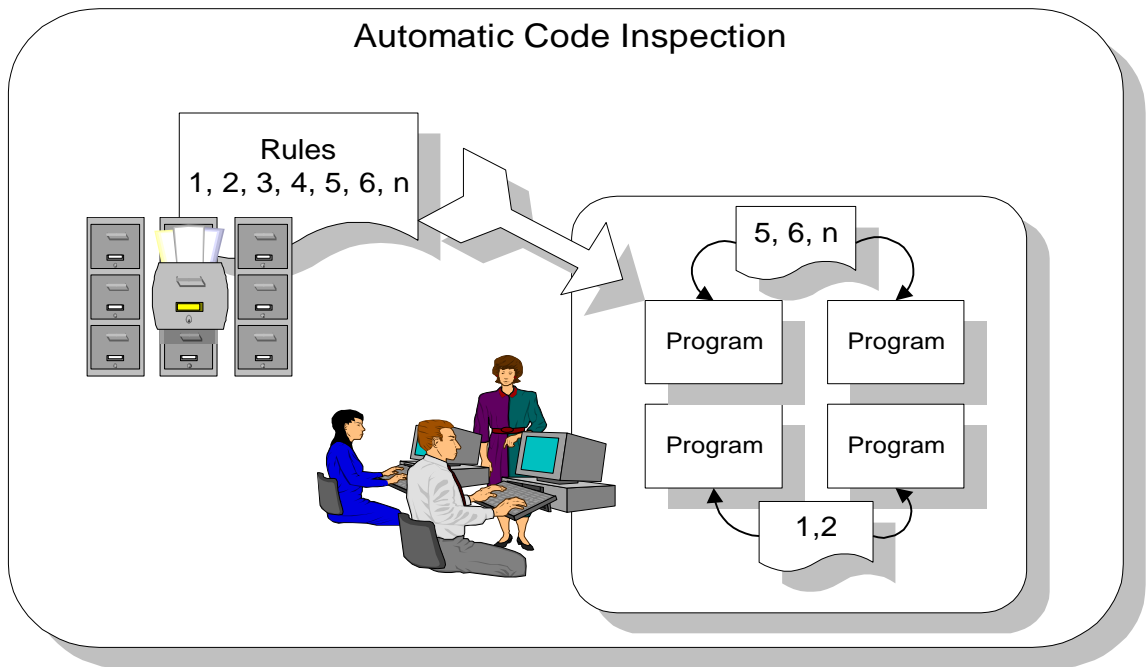
This clearly shows it is economical unfeasible to inspect all the programs. Assuming only 10 percent of the total amount would demand 6 staff per month leaving the 90 percent without revisions.

The idea



With the aim of inspecting the total number of programs, we combined our experience with the automated part of 'Y2K Contamination Control Process', with the one from 'Manual Code Inspections'.

We merged both ideas and wondered if we could check the total number of rules known in all the programs.



We developed what we called '**Automated Code Inspection Process**'.

Automated Code Inspection Implementation Process

The process followed to get ready for the inspection.

It doesn't matter the way you are going to inspect the code, the most important thing is to recognize which elements make your code vulnerable to the introduction of defects.

Depending on the level of accuracy you expect from the automation is the level of sophistication you need from the tool you buy or build.

Source rules identification

The first task to fulfill is the search of all the sources from where you can obtain the rules to be controlled. All the rules should be written and well known by everybody in the company.

Some of the sources identified are the following:

- Code Standards: They're the most important sources. Language std. (Cobol), transactional monitor std. (Cics), database std. (DB2), security issues std., etc....
- Library of reusable modules: Transforming code to allow a migration that requires important code changes often need to maintain modules that are going to be discontinued and new ones that are going to be used gradually. This information is managed by SCM and it's the source of several controls that have to ensure that the new written code is not 'legacy code' and that each program that's taken out of Production environment is returned to it working with the new modules.
- Failure Tracking System: It's another important source. There are a lot of simple defects that could be easily identified. For example, a typical case is the lack of end-of-file checks. It's very important to find the root cause of the defect, as in this way we can develop a rule and verify its compliance in the

rest of the programs, because of the new tendency to create new code by using 'cut & paste' technique.

- Classic Code Inspection ('Fagan style'): Although they're not performed frequently, they always allow us to identify new defects.

Rules development and construction

After identifying all the sources, they should be analyzed so as to define the rules to be controlled. We should avoid ambiguity. Rules like "... *Cobol paragraphs should not be long ...*" or "... *SQL commands should be simple ...*" are useless.

Those rules which can be identified could have different formats. For example:

- Prohibited commands: such as ALTER, GO TO, SELECT *.
- Interface: such as the lack of return code checks.
- Data handling: such as the lack of I/O return codes.
- Exception handling: such as WHEN OTHER clause in EVALUATE statement, AT END clause in SEARCH statement.
- Configuration Management: such as the use of a module that is no longer supported.
- Performance: such as the JOIN of four tables or more.
- Cosmetic: such as how to indent certain commands.
- Naming convention: such as the minimal length of characters in variable names.
- Module size: such as the maximum number of statements it should contain.
- Etc.

A rule violation doesn't mean avoiding a defect, but if violated it makes the code more susceptible to the introduction of new defects during maintenance activities. In some cases there is the assertion about a potential defect, but it can also be a process improvement suggestion or a matter that requires attention. (Remember that the code must be correct, but it should also be understandable, efficient and maintainable).

Each rule belongs to a certain domain. This means that the control of the rule can be performed over the programs that belong to that domain. For example, there are certain commands that are allowed in a batch environment but they're not in an on line environment. Or, naming convention standard has to be followed by all the programs developed in the organization but cannot be demanded to third party developments already built.

Examples of different domains are:

- Batch & On Line
- New programs & Legacy programs
- In-house development & Third party developments
- DB2 / DLI / Vsam files

Once the rules have been developed and their domains identified, they should be validated with the people in charge.

Finally, the rules should be recorded in the Compliance Database (specifying their source and the date they come into effect).

Violated rules detection

The different ways in which rules could be broken within the code have to be evaluated. This task is not easy, and depending on the rules to be controlled, special skills to perform the task will be needed.

The ideal profile of the analyst is:

- Rule domain expertise
- Not less 3 years development experience'
- Survived a bad implementation of an application (Understands the pain of a poor quality application)

Some defects are easy to catch while others are extremely difficult. The language syntax should be well known in order to identify all the situations.

Implementation decision

Taking the decision which rules will be automated with what tool.

Two different kinds of inspection were identified.

- Automated: when a defect is identified without needing any later human analysis.
- Assisted: when a defect is identified but it needs later human analysis. For example, the code can be scanned for certain defined wild card search criteria (pattern-matching technique based) that finds potential defect candidates and then the final decision is made by the analyst through manual intervention. (Some defects can only be found by looking at every line of code and understanding the flow of logic and data within and between programs).

Many rules are controlled through assisted inspection. Sometimes this is because the information obtained from the program is not enough and some environment information is needed to decide if the rule is violated or not. This is the case of DB2 performance related rules. Depending on the number of files of the tables, there will be SQL commands that will be prohibited or not. In these cases, these commands are first detected and later the analyst will decide if there's a problem.

There are other cases when the author's program is needed. For example, this was the situation while searching date related fields in the Y2K conversion project. Variables were analyzed by name, definition and use. If the name of a field referred to 'date', 'year', 'yymmdd' or other such clue, it became a potential suspect. But unfortunately, it is well known that programmers use a great variety of reasonable and unreasonable names for their variables. Correctly finding all true candidates required knowledge about all ways that a date could enter a program and a thorough analysis of everything that could happen to those dates within the program.

So, not everything can be automated. But even less sophisticated tools (developed or bought) are faster and more accurate than performing the same task manually.

To buy or build your own tool is a personal decision that depends on a lot of factors (staff skills, time, money, etc..). The market offers a great variety of tools that help to identify defects accurately which could cause application failure, data corruption or unpredictable results (such as arithmetic overflow, uninitialized variables). Selecting a tool is a hard activity. The difference in product cost is often due to product quality and reliability, vendor experience and level of available support (it's important to pay attention to capabilities rather than price). Depending on the level of accuracy you expect from the tool is the level of technique sophistication you will need from the tool (scanning, parsing, control flow analysis).

Automated Code Inspection Execution Process

A four-step process to execute the inspection.

Candidate programs identification

Several ways to select the programs to be inspected were identified.

- New programs / Modified programs: Every program that is taken into Production environment (either a new or an existing one with enhancements), should be inspected. This is the criteria used to perform the activity daily and to ensure that Production environment is not going to be contaminated.
- Programs belonging to a certain domain: In this case, all the programs that belong to a certain domain are inspected. This criteria is often applied when a rule is discovered and we want to ensure that there are not existing programs in Production environment that violate it.
- Reported as fixed: Each program that is reported as fixed is inspected in order to verify the defect removal and that no new defect has been added.
- On demand: Key projects ask for revisions early in the construction phase of the life cycle in order to evaluate quality and standard compliance during the construction phase (validating code from the very beginning is helpful)
- Random: Programs are often selected at random from Production environment in order to ensure programs remain defect-free.

Inspection execution

Once the set of programs to be reviewed has been selected, the inspection is executed. First, program's domain should be identified so as to perform the appropriate rules. Analysts start the inspection process, running the tools which automate the different rules.

After the inspection has been completed, analysts check the tool-generated results, reviewing defects with the application owner and making adjustments if necessary. Then, only defects (statements that violates the rules) are recorded in the defect tracking DB.

Defect reporting

Defects are reported to the ones in charge. Reporting date and responsible name should be recorded perform later tracking activities.

The report contains the following data:

- Inspection date
- Analyst
- Program name
- Reason for inspection selection
- Incident description
 - Line number
 - Violated rule description
 - Program statement/s that violate the rule
 - Comments

Tracking

A defect is completely removed only when the program is returned to Production environment and analysts verify it has been fixed satisfactory.

Unsolved defects are periodically requested to be fixed.

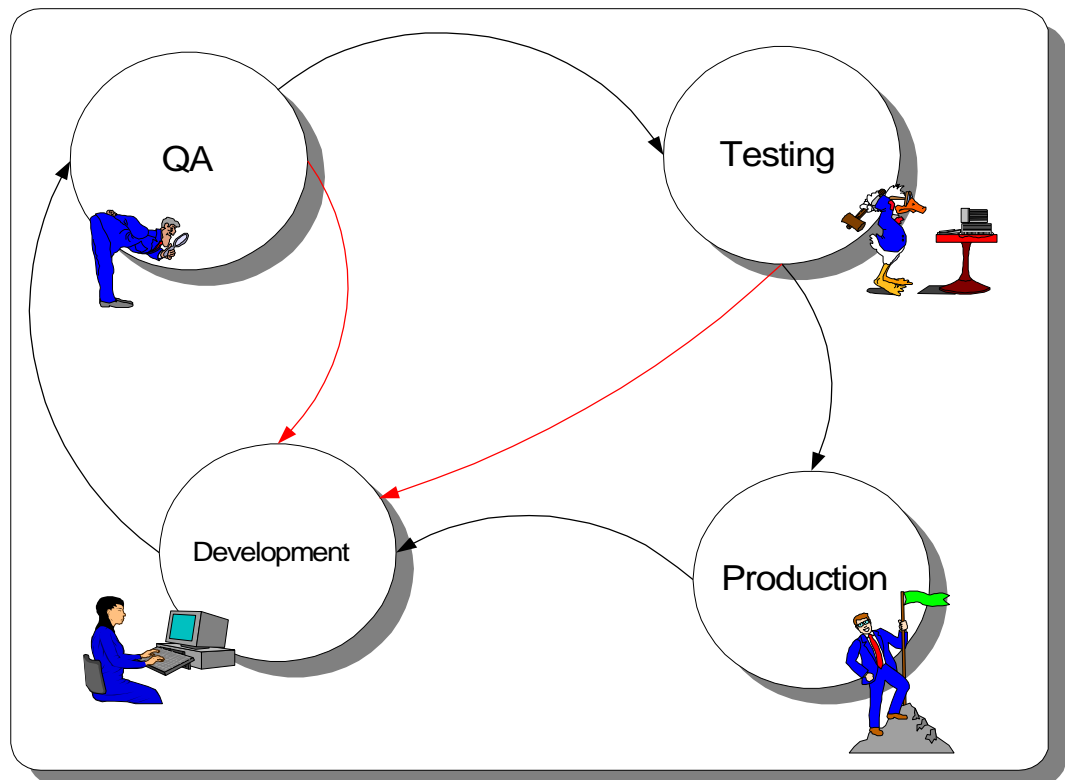
Metrics collections are performed to help the company to control, monitor and assess its software quality goals.

Where it fits in project life cycle

Inspection should be executed before testing, saving time and money.

The Development Cycle

Code is inspected prior to testing stage. All common and identified defects should be removed before a program leaves QA stage. In this way, inspection ensures that fewer trivial defects will delay the application testing, saving time and money (they are found faster and at a lower cost than in the testing environment).



Conclusions

Code Inspection is such a good practice that, even applying it partly, lets you obtain good results.

Inspection is the most useful and cost-effective form of error removal. Everybody agrees consistently in the value of inspections.

We experimented an alternative to classic inspection (Fagan style). The Y2K challenge and our today's schedule driven projects force us to find another approach that lets us reach inspection well known benefits. But this kind of inspection doesn't replace 'Fagan style' one. It complements it.

Detecting certain classes of defects in all the existing programs instead of searching for all the possible defects in one program gave us very satisfactory outcomes.

This is our experience. We know that the process has weaknesses and we have to continue improving. But we gave an important step and we came to the conclusion that there is no excuse for not doing inspections. We hope you agree.

Automated Code Inspection

A practical approach to improve code inspection efficiency

Banco Rio de la Plata work experience

Marcelo Dalceggio

Alvaro Ruiz de Mendarozqueta



November 1999 ~ Argentina



Presentation Overview

- Who we are
- Y2K Experience
- Manual Code Inspection
- How the idea originated
- Automated Code Inspection Implementation Process
- Automated Code Inspection Execution Process
- Where it fits in project life cycle
- Conclusions & Achievements



2



Who we are

- Banco Rio de la Plata is one of the major private banks of Argentina and belongs to Banco Santander Central Hispano
- IBM mainframe based core systems
- COBOL & Assembler, CICS, VSAM, DB2
- 250 branches executing one million transactions a day and providing universal banking services
- The goal: Reduction of costs in the software development and maintenance



3



Y2K Experience

- Y2K Contamination Control
 - Windowing technique solution
 - Code revision to detect possible violations to the Y2K fix
 - Few Y2K Compliance rules to be applied in all the programs
 - Code-assisted inspection process
 - Very satisfactory outcomes



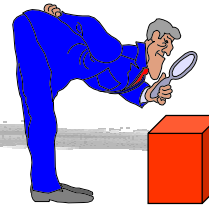
4



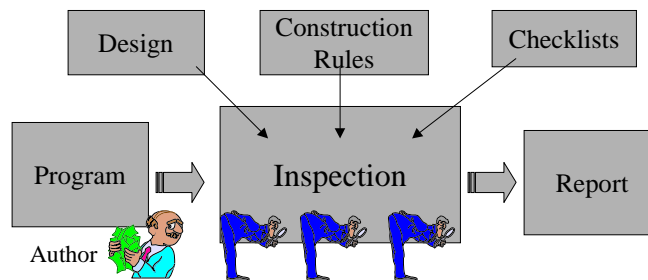
Code Inspection

- There's no doubt of its advantages
 - Defects are detected before the testing process, saving time and money
 - Identifies defects that testing misses
 - Gives complete code coverage
 - Finds root cause defects (no symptoms)
 - Additional advantages
 - Improvement opportunities
 - Good & Bad practices are finally learned

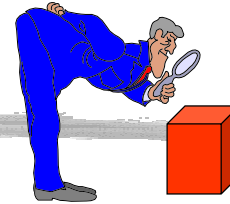
Manual Code Inspection



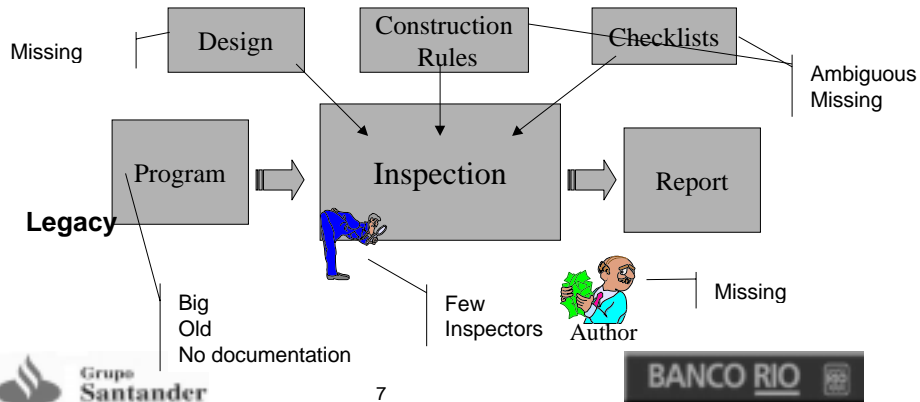
- Code Inspection formal outline



Manual Code Inspection

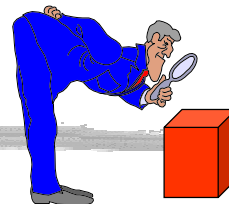


Everyday problems



7

Manual Code Inspection



Environment difficulties

Staff Size

- | $900 \text{ pgms} * 1\text{KLOC/pgm} = 900.000 \text{ LOC per month}$
- | $900.000 \text{ LOC} / 100 \text{ LOC/hour} = 9.000 \text{ hours}$
- | $9.000 \text{ hours} / 160 \text{ hours/staff} = 56 \text{ staff}$

Inspector candidates

- | 'Key projects' demand best resources
- | Without formal training
- | Not used to team work

Manual Code Inspection



Environment difficulties

Programs

- | COBOL & Assembler, 1KLOC average size
- | Lack of program documentation
- | 10/15 year-old programs
- | No design documentation

Inspection rules

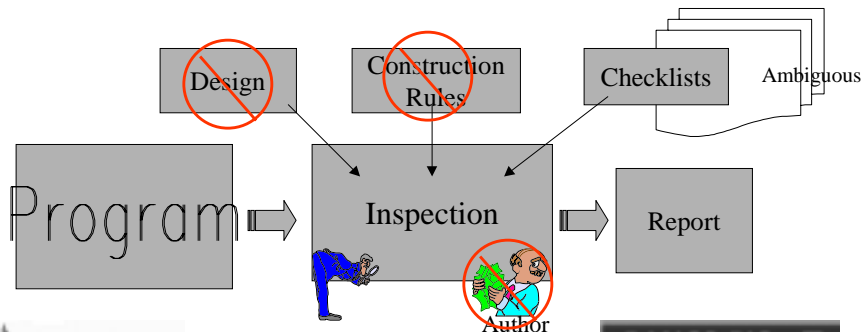
- | Too many rules to be taken into account
- | Ambiguous
- | Application domain rules not well defined



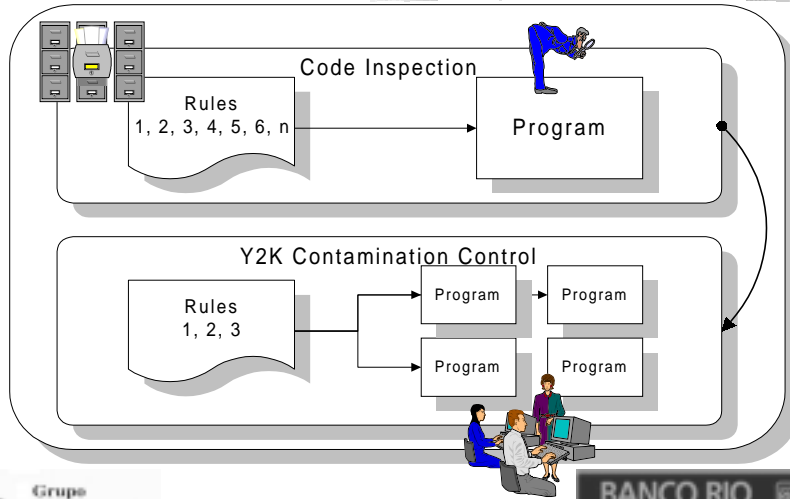
Manual Code Inspection



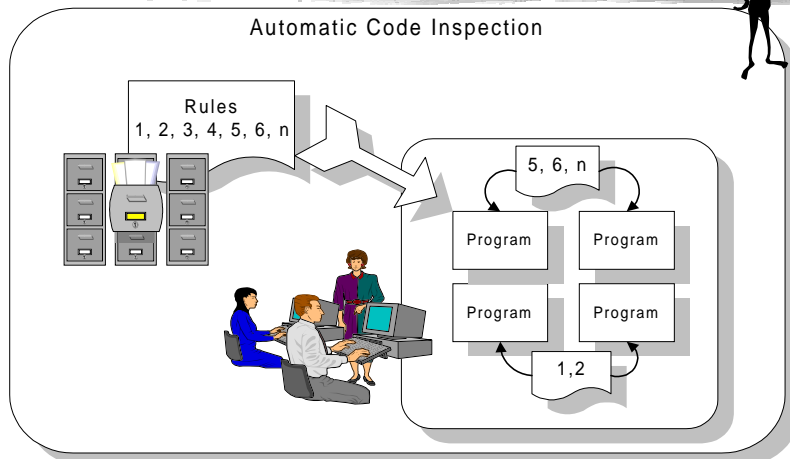
Most common scenario



How the idea originated

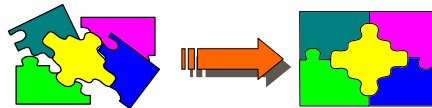


How the idea originated



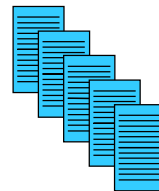
Automated Code Inspection Implementation Process

- Process applied
 - Source rules identification
 - Rules development & construction
 - Violated rules detection
 - Implementation decision



Automated Code Inspection Implementation Process

- Source rules identification (I)
 - Code Standards
 - SCM
 - Library of Reusable Modules
 - Existing /Discontinued Modules
 - Good practices
 - Failure tracking system
 - Classic Code Inspections (Fagan style)
 - Development area 'feedback'



Automated Code Inspection Implementation Process

- Rules development & construction (II)
 - Identify rules from each source
 - Identify application domain from each rule
 - Avoid ambiguities
 - | "... Cobol paragraphs should not be long ..."
 - Validate rules with those in charge
 - | Sponsor's support is very important
 - Record rules in the Compliance DB



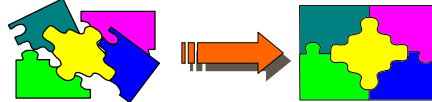
Automated Code Inspection Implementation Process

- Violated rules detection (III)
 - Discover different ways in which rules could be violated within the code
- Implementation decision (IV)
 - Evaluate detection method
 - | Scanning, parsing, control flow analysis, etc..
 - Define automated/assisted detection
 - Tool: Buy or build ?



Automated Code Inspection Execution Process

- Process applied
 - Candidate programs identification
 - Inspection execution
 - Defect reporting
 - Tracking



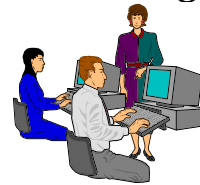
Automated Code Inspection Execution Process

- Candidate programs identification (I)
 - Criteria definition
 - New & Modified
 - Belonging to a certain domain
 - After or Before a given deadline
 - Reported as fixed
 - On demand
 - At random



Automated Code Inspection Execution Process

- Inspection execution (II)
 - Program domain checking
 - Rules compliance verification to each identified domain
 - Record defects in the defect tracking DB

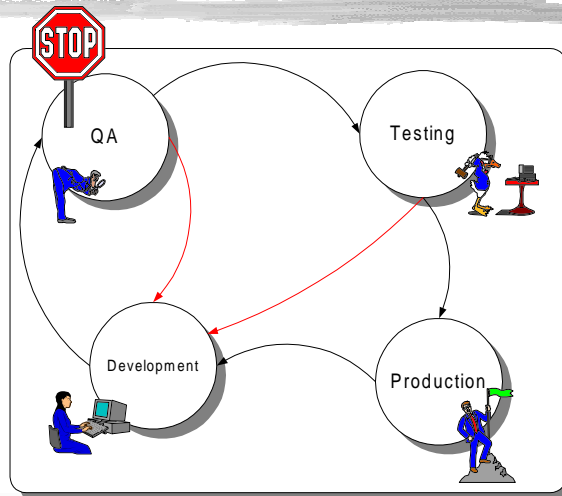


Automated Code Inspection Execution Process

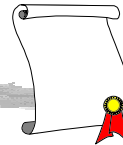
- Defect reporting (III)
 - Inform the ones in charge
 - Record the report delivery
- Tracking (IV)
 - Request unsolved defects
 - Verify the ones solved
 - Metrics collection



Where it fits in project life cycle



Conclusions



- This kind of Inspection...
 - is a good alternative in today's schedule-driven projects
 - is more scalable than manual inspections
 - More rules and more programs
 - Few resources
 - Adding rules do not affect inspection efficiency
 - does not require 'special' skills
 - **does not** replace 'Fagan Style' inspections

Conclusions



- Once a new type of defect has been detected it is possible to...
 - prevent it from happening again, and
 - get rid of it from the existing programs
- Inspection ensures fewer trivial defects will delay the application testing
- Code quality & reliability metrics could be defined

Achievements



- 157 rules identified
- 750 KLOC per month with 3 QA reviewers
 - Includes full Automated Code Inspection Process
- > 1650 defects detected
- Reporting 'on demand' for different critical projects
- Reporting 'on demand' for third party developments



Software Project Evaluation as a vehicle for SPI

Hanna Luden

Getronics Software Solutions

Abstract

Once an organization begins with the software process improvement adventure, a long-term path is entered. Already during the assessment software professionals and their managers are encouraged to discuss and evaluate their current practices, with their pros and cons in order to improve them. The long lasting silence following the assessment activities, due to the need to carefully analyze and plan the improvements, forms a risk of losing the gained momentum.

This paper discusses project evaluation as a simple - almost obvious - tool that can effectively help preserve the gained momentum. It is also a means to spread awareness for the need for change and to exchange information about best practices.

We discuss our experience with qualitative project evaluations (QPE). QPE must be carefully planned, carried out following a script, and documented in the broadly accessible *project evaluation database*. Thus, the evaluation serves as an important vehicle for *continuous improvement*, providing the organization with direct information about its actual practices. It offers early evaluation of relevant aspects of the software process, helps set priorities for improvement actions, allows monitoring the effects of already introduced improvements and of culture, and bridges the gap between software professionals and the software process improvement program.

Introduction

Software process improvement (SPI) programs typically begin with an assessment of the organization. People with different roles in the software organization participate in the assessment and generate ideas for improvement and change based on their experience, knowledge and insight in the current practices. The assessment is usually the start of a long lasting Software Process Improvement Program. After the assessment has been carried out and the results have been summarized and presented, plans for improvements, related to the business goals, need to be set and realized.

Careful SPI-planning costs time, and before the improvements show results, even more time elapses. Yet, it is of major importance to preserve the momentum initiated by the assessment. The challenge for every SPI program is to gain and maintain that momentum.

This paper discusses a technique for qualitative project evaluation (QPE). QPE can be introduced quickly, and does not demand much effort or time while allowing high leverage. If planned carefully and carried out correctly QPE can become an effective organisational SPI tool and reduce the risk of losing the painfully gained momentum.

Objectives

Having identified the risks of a long silence from the SPI-initiative after the assessment, we have searched for ways to preserve the existing enthusiasm and involvement. A working group was appointed, and the following objectives were set:

- Introduce an instrument for *continuous improvement*, in line with awareness and will to act gained during the assessment, and maintain the visibility of the SPI program in the organization.
- Use best practices: allow projects effective re-use of (work) products, tools, methods and techniques from other projects. In particular, identify potential ‘SPI-instruments’ and ‘SPI-products’ generated by the project which could be upgraded and offered to the organization as a whole.
- Provide a platform for individual employees to express their opinions and share their experiences in a *constructive and open manner* for their own learning as well as for the organizational learning.
- Provide the management with information about the practitioners’ experiences and opinions.
- Institutionalize an instrument for communicating these experiences to the management (and back), and stimulate a culture of openness about those issues.
- Gain insight into the effectiveness and efficiency of the current processes and practices, and in particular into the (new) improvements, thus monitoring the SPI-program itself.
- Provide the SPI-program with direct information about the practitioners’ experiences and opinions regarding the SPI program and thus enabling eventual fine-tuning (or bigger changes if necessary) of the SPI-program. In other words, indirect evaluation of the SPI-program itself.

Qualitative Project Evaluation

Project evaluation is no news. Which organization does not have an evaluation-form to be filled-in (after project completion, sometimes at other moments)? Yet, in how many organizations a serious evaluation is actually carried out? Does everyone participate, or is it a burden to the project manager which prevents him or her from neatly rounding up the project, while the horrors of the new project already take 150% of their time? And what do organizations do with the outcome of evaluations?

Qualitative project evaluation (QPE) was introduced to meet the objectives mentioned above. We have chosen to make the project evaluation a *defined process* within the software development process. At the same time we believe that it should become a second nature, and be used in *all* relevant situations. Once people have experienced its added value, they do not let go!

In given moments in a project’s life (not just after completion!), we take as a team a break - time-out - to reflect¹. The QPE is carried out in a meeting, which is thoroughly prepared and planned, and which is carefully documented.

Everything may be evaluated, and evaluation can (and should) have unexpected results. After all, it is not the obvious and common knowledge we are after.

¹ The *qualitative* evaluation is carried out *next* to the obvious collection and evaluation of quantitative information. QPE is a change-management instrument. Among other things, the qualitative information provides more insight into - and a better understanding of - the collected data.

Qualitative Project Evaluation: The process

Having recognized the flaws of current state project evaluations, we looked for a different approach. An effective evaluation needs good preparation, and a good *process* underlying it.

Our Quantitative Project Evaluation process has the following stages:

Preparation:

- Determine *motivation* for the evaluation (end of project, end of phase, x months further, change of course, change of leadership, etc.),
- Determine *objectives* and select the *focus point(s)* for the evaluation (process, methodology, use of a certain tool, a Key Process Area, external factors, etc.),
- *Tailor* the general script to suite the objectives and focus points.
Other relevant parameters are the project's stage, the number of participants, etc. *Review* the script with a project member.

Execution:

- This is *the actual evaluation session*, led by a moderator assisted by a scribe. Both are *external* to the evaluation team, and need only to focus on the process. As mentioned above, the meeting is carried out according to the script. During the session the project members discuss the successful aspects and the problems, they analyze the causes, and generate ideas about improvement actions needed on individual, project and organization levels.
- Determine *follow-up actions* to be taken. This is a very important point. The participants can use the evaluation's output to improve their own work, and if the evaluation is not at the end of the project, the project can and should benefit from it. Many aspects are relevant for the management and for the organization as a whole, and it is important that the participants *and* their colleagues see that follow-up actions as a result of the evaluation actually take place.
- *Evaluate the meeting*: a short evaluation at the end of the meeting, (a questionnaire to be filled in). We use this to improve the QPE process.

Documentation and follow-up:

- It is the scribe's responsibility to document the meeting in such a way that the participants recognize their input. At the same time the evaluation (as a group's result!) should be understandable and relevant for others in the organization, while individual-inputs *should not* be identifiable to others.
- The participants review the evaluation-document.
- The evaluation is made available to the organization as a document, and in a database (the so-called *project evaluation database*).
- The follow-up actions defined should be carried out; the project-evaluation coordinator tracks the follow-up actions.
- Provide feedback about the outcome of the actions to all participants.

The Project Evaluation Database

This database, accessible to the organization, contains the relevant outputs of all the evaluations. It also contains (pointers to) *quantitative information* about the project (such as plans vs. actuals), and a so-called '*project-profile*'. The project-profile is the characterization of the project in terms of size (small, medium, large, very large), form (fixed price, fixed date, etc.), technical profile (platform, methods and tools used, programming languages, etc.), and other characteristics.

The *Project Evaluation Database* serves several purposes:

- It is a simple means of communicating about projects as well as about the outcome of project evaluations to a wide population.
- It serves as a simple information provider for new members in teams and for new people in the organization and provides a flavor of the organizational culture.
- It can be used as a knowledge-base.
- Also it allows insight into trends in the organization's behavior, and about its strengths and weaknesses. The information is relevant for prioritization and decision-making regarding improvement actions (next to periodical SQA-reports, change requests, etc.).

Qualitative Project Evaluation: some hints

Be critical. It is easy to say what everybody else should have done better or different. External groups or individuals, existing procedures and the like, can obviously be addressed. But, what could *we*, as individuals and as a team, have done to stimulate better results? Make people realize *they* can influence the process by changing their own behavior.

Communicate the evaluation output. Remember that the audience is not only the participants, but also others who know little about the project's ins and outs. Make sure the evaluation is understandable to all!

Make sure that **feedback** by (line) management and other external groups is provided. The biggest risk is that all the findings remain within the evaluating groups, put on paper, and little or no improvement actions are taken or made visible. The management's *actions*, as well as a positive and open attitude *is a prerequisite* for the sustaining success of evaluation activities. There is no bigger demotivator than the feeling of not being taken seriously, especially by one's superiors.

Do not forget that evaluations only *identify* needed change. The changes still need to be realized! **Resources** - both financial and human - must be made available in order to make the changes really happen.

Actual actions should be agreed upon *before closing the evaluation session*, to be carried out by the participants, in order to ensure that 'things really get done'. When we go back to our daily activities, we tend to forget. A 'project evaluation coordinator' should remind people about their commitments made during the evaluation session. Defining deadlines for actions is very helpful.

The *Project Evaluation Database* need not be too ambitious. Start simple, and add key words and (extra) structure while going. Use the limited resources for evaluation and communication. Our experience is that carrying out the evaluations is the best promotion. Individuals are stimulated to take more initiatives. Make sure, though, that the database *can* be extended in a controlled and economical manner.

Stimulate **early phase evaluations**, long before the project is finished. The project members can then directly apply the evaluation's results in their work. This is an important cultural stimulator, seeding the notion of evaluation in peoples' minds, rather than in the process only.

Moderators must be **well trained!** Moderating a session seems deceptively simple, but it is not. The effectiveness of the meeting and the level of the results depend on both a well-prepared and carried out process, and on the critical attitude of the moderator.

Conclusions

A carefully prepared and planned project-evaluation, carried out in meetings to discuss qualitative aspects, is an excellent instrument to create an open and constructive culture and to support continuous improvement. Commitment from people in different functions and levels in the organization is effectively gained. Organized evaluation meetings, following a script, carefully documented and communicated, significantly contribute to Software Process Improvement programs. The evaluations create a platform to discuss issues, which are experienced as 'the real problems'. The good practices of projects are offered to others in the organization.

Project evaluations provide the SPI program continuous insight into the actual practices and 'hot issues' in the organization. They allow monitoring the effect of improvement actions and provide information for prioritization and planning of improvements. It is an ideal vehicle for creating and sustaining an open culture needed for the learning organization and for change.

Documenting the output and making it broadly available – in combination with the quantitative and technical project information - provides the organization with simple, directly accessible knowledge base for individuals, projects and managers.

Visible management involvement and commitment is essential for success, and contributes significantly to the success of the SPI-program as a whole.



Software Project Evaluation
as a vehicle for
Software Process Improvement

Hanna Luden
Getronics Software Solutions

Content

- **Context;**
- **Challenges;**
- **Project evaluation;**
 - ⇒ What;
 - ⇒ When;
 - ⇒ The process;
 - ⇒ Example;
- **The Project Evaluation Database;**
- **Our experience with project evaluation;**
- **Conclusions.**



Context

SPI Programmes take long to deliver...

- Awareness;
- Decision making;
- Preparations;
- Assessment;
- GAP analysis;
- Improvement plans;
- Task forces;
- Action planning
-

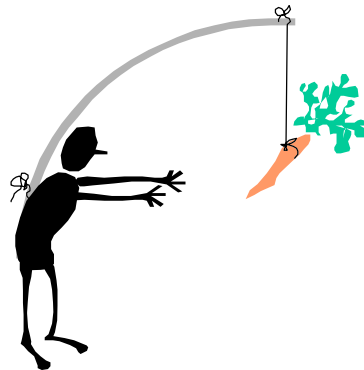


Getronics

Context (cont.)

SPI Programmes take long to deliver ...

- Getting everyone on board takes long, and costs a lot of effort;
- People have seen prior improvement actions fail;
- “Yet another improvement action????”



Getronics

Context (cont.)

*Sustain momentum and gained commitments:
something small, effective, and quickly
realisable is needed!*

Getronics

Challenges

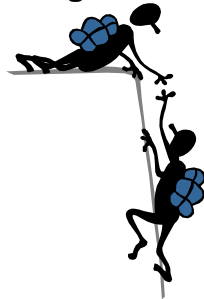
- **Sustain the momentum gained with the assessment;**
- **Maintain visibility of commitment;**
- **Provide a platform for the exchange of knowledge, experience, ideas and tools;**
- >>



Getronics

Challenges (cont.)

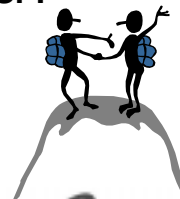
- Stimulate individual and organisational involvement and learning;
- Provide management insight into professionals' views;
- Reuse existing knowledge, experience, ideas, etc.;
- >>



Getronics

Challenges (cont.)

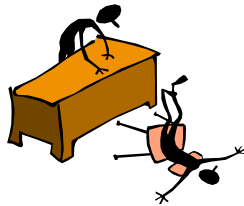
- Gain insight in effectiveness current processes;
- Create input for prioritisation of improvement actions;
- Create an instrument for ongoing improvement actions;
- Identify potential SPI-instruments and SPI-products;
- Stimulate open culture.



Getronics

Project evaluation

- Project evaluation is *not* new;
- Project evaluation is *not* the Solution to all problems and risks;
- Project evaluation is *very effective in helping the organisation grow*;
- Project evaluation must be *institutionalised* for best results.



Getronics

Project evaluation: What

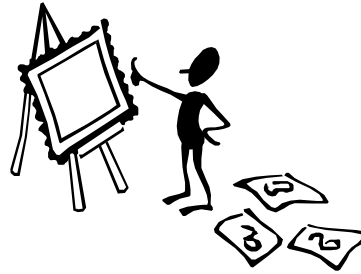
- Project evaluation is a *defined process*;
- *Lessons-learned* for individuals and groups;
- *Qualitative* ('soft') aspects in focus;
- Generate and collect *common knowledge and experience*;
- >>



Getronics

Project evaluation: What (cont.)

- **Monitor** the software process, work, achievements, faults and weaknesses;
- **Every process** can be evaluated and learned from;
- Search for the *unexpected!*
- **Do it!!!**



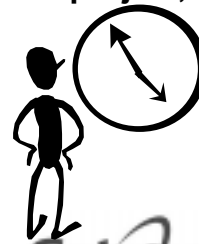
Getronics

Project evaluation: When

- **Classically: at *project end***
BUT the *group* cannot directly apply results;
- ***During the project*** at set moments
(milestones, change of course, change of leader, some crisis, etc....):

Findings are *directly applicable* for the project;

⇒ **At least twice in the life of a project, or every 4 months.**



Getronics

Project evaluation: The process

- The process evaluation is carried out during a special *meeting* of the project group;
- The meeting is facilitated by a *moderator* and a *scribe*, external to the project group;
- The process is defined, supporting facilities are used;
- Good preparation ensures effective results;
- >>

Getronics

Project evaluation: The process (cont.)

Evaluation process phases:

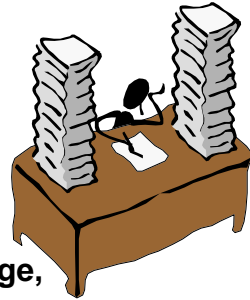
- step 1: Preparation;
- step 2: Execution: the evaluation meeting;
- step 3: Documentation and follow-up.



Getronics

Step 1: Preparation

- Determine *motivation* for the evaluation;
- Determine *objectives*;
- Select *focus points*;
- Tailor the general script
(motivation, objectives, project stage,
number of participants, etc.);
- Review the script with project representative.

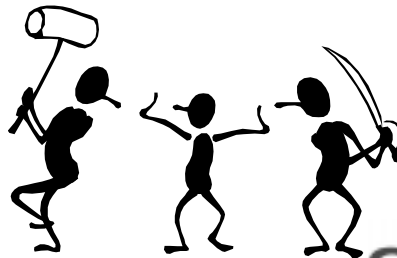


Getronics

Step 2: Execution

This is the actual evaluation meeting:

- Takes typically 4 hours;
- Participants: all project members;
- External moderator+scribe: focus on *process* only;
- All project members are actively involved;
- Make the meeting enjoyable;



Getronics

Step 2: Execution (cont.)

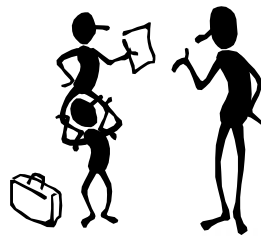
The evaluation meeting phases:

1. Evaluate:

⇒ Good, bad, causes, how could **we** have acted/ act differently, what could others do differently and how can we achieve this?

2. Determine follow-up actions (what, who, when);

3. Evaluate the meeting.



Getronics

Step 3: Documentation & follow-up

• Scribe documents the meeting:

- ⇒ Recognisable for participants;
- ⇒ Understandable for outsiders;
- ⇒ Individual input not traceable.



• Participants review;

• Evaluation made generally available:

Project Evaluation Database;

• Follow-up actions carried out (and tracked!), including feed-back to the project group.

Getronics

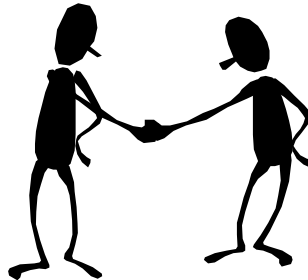
Project evaluation: An example

- **Motivation:** Change of 2 major requirements;

- **Objectives:**

- ⇒ Understand customer expectations;
- ⇒ Better requirements management;
- ⇒ Grip on commitments to the customer.

- >>

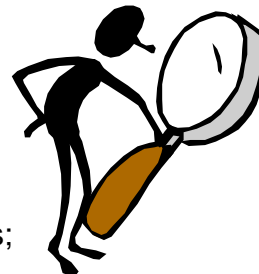


Getronics

Project evaluation: An example (cont.)

- **Focus:**

- ⇒ Project planning;
- ⇒ Requirements management;
- ⇒ Communication with customer.



- **Project group size:** 10 members;

- **Script** is tailored and reviewed with 2 members of the project;

- **(Dinner** scheduled after evaluation meeting);

- >>

Getronics

Project evaluation: An example (cont.)

- **Follow up actions agreed upon:**

- ⇒ Project leader schedules a meeting with the customer to better understand the customer's business goals; all project members attend;
- ⇒ Project estimates will be reviewed by entire project group;
- ⇒ Account manager + line manager will be presented the project's achievements monthly;
- ⇒ >>



Project evaluation: An example (cont.)

- **Follow up actions agreed upon:**

- ⇒ "Risk management" and "Requirements management" added to project meetings agenda;
- ⇒ Hans is appointed the "requirements manager";
- ⇒ Thursday lunch hour introduced for project group, customer & future users.



The Project Evaluation Database

- Easily accessible for the organisation

- Contents:

- ⇒ Project profile (technical and other);
- ⇒ Quantitative information (metrics);
- ⇒ Evaluation(s) results;
- ⇒ Any other relevant information.



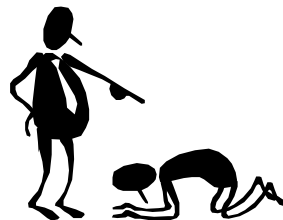
- Owner: the Project evaluation *co-ordinator*.

Getronics

Our experience with project evaluation

- “Its fun, and we learn a lot”;
- A platform to discuss the `real issues`;
- “What will be done with the results of the evaluation? Will something actually change?”
 - ⇒ Feed-back by management is vital!

- >>



Getronics

Our experience with project evaluation (cont.)

- Follow up must be taken seriously, people want to feel their opinions and ideas count;
Actions must be noticed: *Visibility*;
- Many follow-up actions cost little and have a big leverage;
- Reserve *resources* to carry out actions;

• >>



Getronics

Our experience with project evaluation (cont.)

- Projects could learn and use a lot from other projects;
- *The Project Evaluation Database*: start simple!
- Early phases evaluations have more impact, the project group is in 'control';

• >>



Getronics

Our experience with project evaluation (cont.)

- Moderator's role is important; Moderators must be well trained (it seems so simple...);
- The critical moderator sharpens the conclusions.



Getronics

Conclusions

- Project evaluations help *understanding and improving* the way we work as individuals, groups and organisations;
- The *projects evaluation database* allows the organisation insight into its performance, the hot issues and is a `knowledge management` instrument;
- Project evaluations must become part of the organisation's *culture*;
- Project evaluations help *reduce risks of SPI* initiatives;
- >>

Getronics

Conclusions (cont.)

*Take project evaluations seriously or
don't start with it!*

Getronics

Questions



Getronics

Where to find us

Hanna Luden
Getronics Software Solutions
P.O Box 22678
1100 DD Amsterdam
The Netherlands

Tel +31 20 430 6360

fax +31 20 430 6032

e-mail: H.Luden@getronics.com



How to implement structured testing in narrow time boxed projects: Rushing from chaos to structure?

Lieven Schouwaerts

Gitek nv - *interaction through software*

St. Pietersvliet 3, B-2000, Antwerpen, Belgium

Tel: +32 3 231 12 90 - Fax: +32 3 226 10 83

E-mail: ls@gitek .be

1 Introduction

With this paper, I would like to share my experience in introducing structured testing in narrow time boxed projects.

1.1 Background

The paper is based on the introduction of structured testing at a large Belgian bank. This introduction took place at the time this bank merged with another Belgian bank. The merger led to a lot of relatively small projects with a lot of interfaces. Over different releases various projects had to be released simultaneously. It concerned applications of different types on a variety of platforms.

Extra risk factors were:

- A lot of resources shared by the different projects and the different releases.
- Due to the merger:
 - ◆ people with a different “culture” had to work together;
 - ◆ there was a knowledge problem;
 - ◆ there was a terminology problem;
 - ◆ there were structural changes within the organisation;
- Fixed time boxes (release dates) due to dependencies and commitment to the business;
- Different usage and management of the IT environments.

1.2 Rushing from chaos to structure?

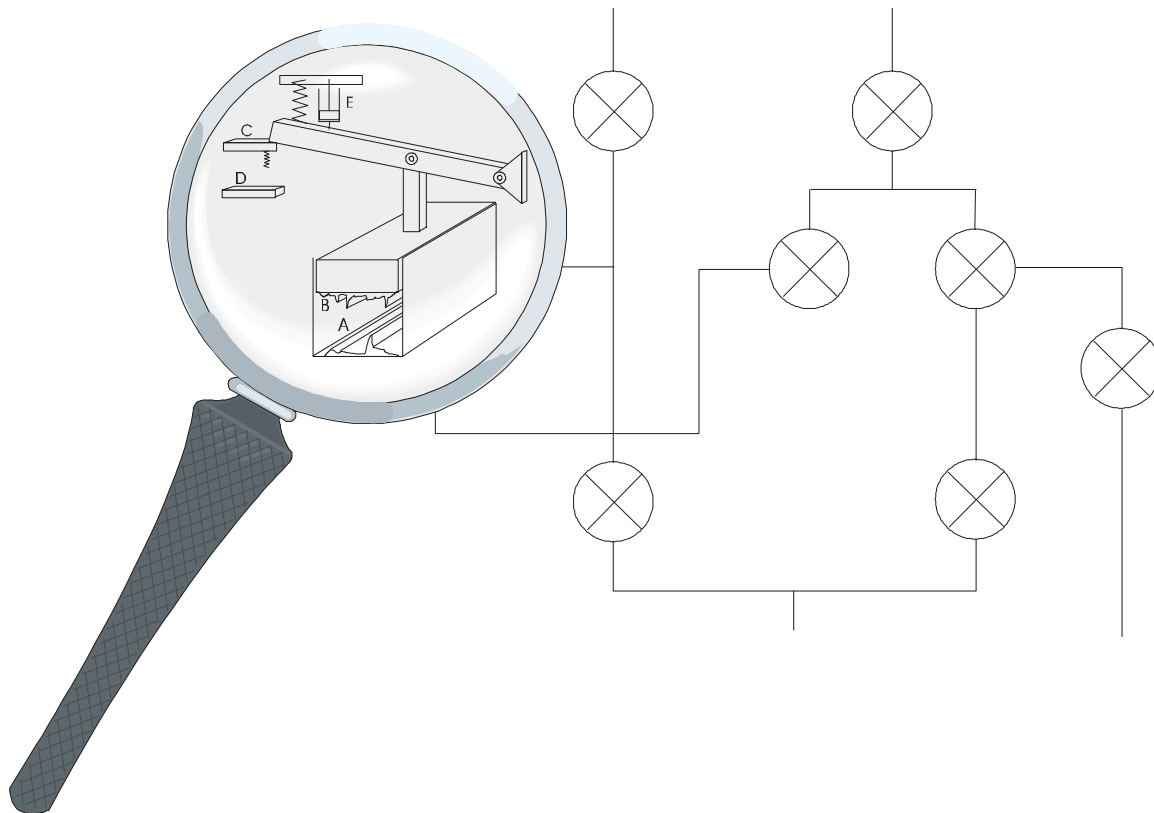
Although the title might suggest so, it's not about a rush from chaos to structure because:

- Rushing is not good: More haste less speed. Experience learns that hasty changes in company processes usually result in uncontrolled initiatives, waste of time and frustration rather than in quality improvement of the end product;
- Start from chaos? No! : The existing approach for developing applications already contained a certain structure. The V-model was applied to this structure when introducing the new test approach;
- Structure? Yes! : We tried to structure things as effective as possible, based on the available time and resources.

2 Possible approaches

When you want to structure your test process, but haven't got time or resources to do it in one go, there are a lot of approaches one can follow.

To explain why certain approaches were or were not chosen, I drew the following scheme, representing a flow of activities and products through a network of which each part can have different properties. The scheme doesn't include all dependencies and properties of actions and products that emerge from a project, it only includes the necessary elements to explain the following reasoning.



The different elements represent the following properties:

- The size of the section (A) determines the speed. A larger size represents a faster process but doesn't tell you anything about the quality;
- The roughness (B) of the surface determines the smoothness of the activity and the quality of the resulting product;
- The overlap of the small plate (C) determines the possibility of something going wrong. The smaller the overlap, the easier things can go wrong. This is not equal to the risk because the overlap doesn't tell you how bad things will be when something goes wrong;
- The position of the stop (D) determines the end severity or final damage when things go wrong and no corrective actions are taken in time. This position usually determines the priority when people have to decide about risk factors in a haste, although that does not always lead to the right decision;
- The spring and plunger (E) determine how fast things will change when something goes wrong and how easy it will be to fix it.

In fact (A), (B), (C), (D) and (E) together determine the total risk that is linked to a process because they all influence the end quality or timing of the project, and those are the two things that usually matter when one discusses the success of an activity.

2.1 *We didn't choose to Structure one part of the process*

You could start to structure only the infrastructure of your process (setting up test environment and data) or only the test techniques, organisation or planning, but the “unstructured” parts of your process will probably undo the effort spent on the one part.

Putting a quality product through a rough section (B), will lower the quality and create a stress situation between the ones who deliver the quality product and those who “destroy” it.

Using low quality input for a good quality activity will give poor quality results and usually people don't put

Widening a section (A) of an activity that is used as input for in a smaller section (A') will not speed up the entire process, it only enables you to get that specific activity off the critical path.

2.2 *We didn't choose to structure everything a bit*

This is focusing on the company objectives only. An equal focus on project objectives and objectives of individuals as well, has more chance of surviving the introduction of a new methodology.

The differences in maturity or quality of the different parts remain, so the losses caused by these differences stay the same. This causes the same people to be responsible for the “less than average” parts of the process over and over again and can result in even worse functioning of the concerning unit, enlarging the quality gap and worsen the situation in stead of improving it.

Especially in parts where there's a lot of work to do, it will take a long time before results will show and people will notice the effect of the effort spent on structuring. Meanwhile negative feedback of depending components will remain. This usually results in giving up.

2.3 *We chose to set priorities according to risks, objectives and current level of maturity*

This enabled us to create a balance in the maturity level of the different processes and align them in an efficient way.

Concentrating on bottlenecks and weak links in the chain of activities immediately results in improvements. It is not only advantageous for the concerning part, but for all parts depending on it.

It also allows using the limited available time and resources as efficiently as possible. This is very important in small time boxed projects, preventing to get the “drop in the ocean” effect.

3 **Extra activities: going and growing concern**

Next to the introduction of the new test approach, a lot of other activities (existing and new) have to take place. Focusing too much on introducing what's new may result in neglecting other activities.

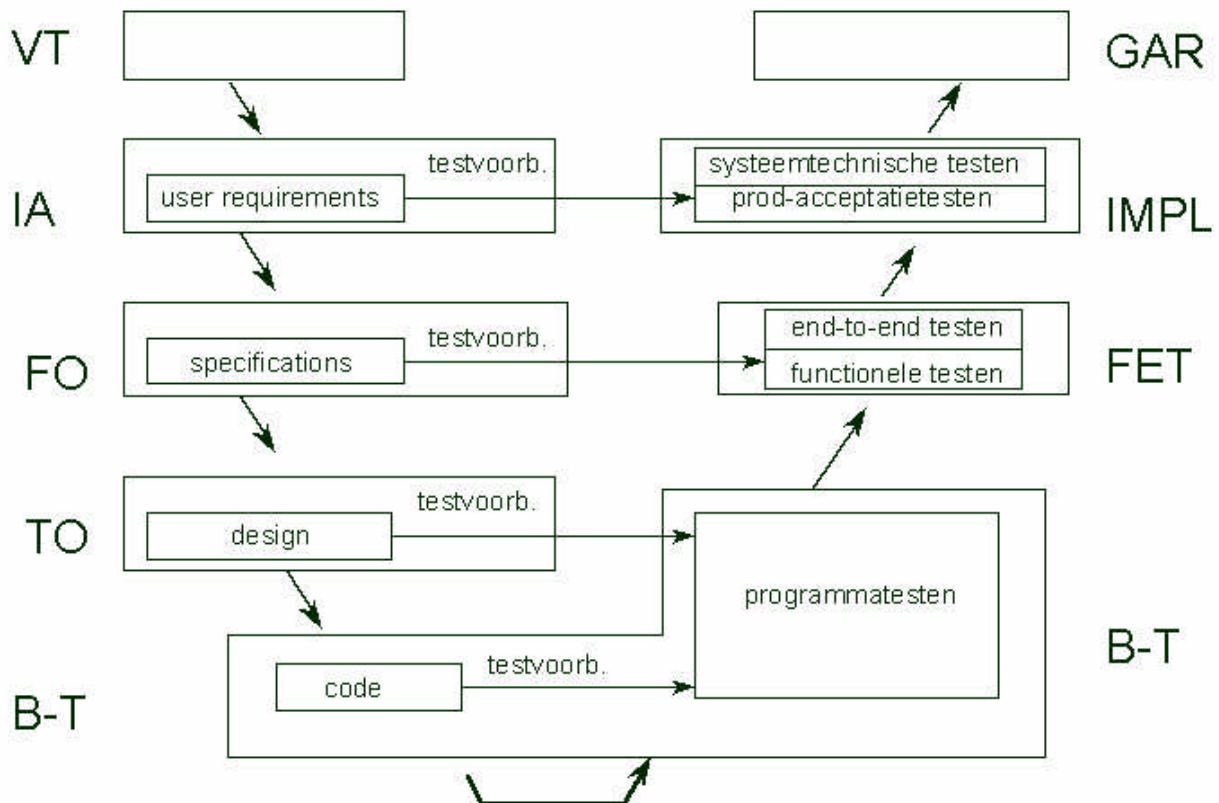
- “Business as usual” activities: maintenance, customer support, fixes, ... ;
- Changing of roles and responsibilities due to the merger;
- Y2K testing;
- Euro follow-up;
- QA: The introduction of structured testing had to fit in the Quality Assurance program;

4 How to bring this into practice?

4.1 Define priorities for the general approach

In our case the following priorities and objectives were defined:

- Changing the existing waterfall model into V-model (essential).



VT: Preparation

IA: Information analysis: defining user requirements; test preparation for acceptance testing

FO: Functional design: defining specifications; test preparation for functional and end-to-end testing

TO: Technical design and test preparation for unit testing

BT: Coding and Unit testing

FET: Functional and end-to-end testing

IMPL: Acceptance testing and Implementation

GAR: Maintenance

- Focusing on functional and end-to-end Testing (FET), because this part has got the lowest maturity compared to the other test types;
- In first instance, the scope of the new methodology is limited to the merger projects;
- Project leaders are supported by test consultants who help to translate theory into practice. These consultants are dedicated to one Bank Program (a group of projects linked to a certain bank product or activity). These consultants should have good knowledge of both structured testing and the concerning financial activities (these white crows are still hard to find). All testing activities, including all preparation is done by project members and business personnel of the concerning application.

4.2 Set up the new methodology

Different test types can be described and linked to the V-model, for each test type a life cycle has to be followed. For functional and end-to-end testing the following phases were described:

- Test Plan;

- Test Requirement Hierarchy (TRH);
- Test Design;
- Test Cases;
- Test Procedures;
- Test Execution;
- Defect Tracking;
- Test Report.

A toolkit with templates and examples was set up to help the project leaders save valuable time and to assure consistency between the different projects.

4.3 *Sell the new methodology*

Just handing over instructions with new procedures to project leaders doesn't work. In our case presentations were given to all project leaders, technicians, etc. to create awareness and to make them familiar with the new methodology.

One or two test consultants were assigned to each bank program. One bank program contains 20 to 40 projects over different releases. Test consultants personally contacted every project leader within their program to discuss the specific approach that could be applied in their project, using those parts of the toolkit that led to the best efficiency/effort rate.

Because the project leaders have very different backgrounds (due to the merger), a personalised approach is required. With this personal approach almost each project leader can be convinced of the added value. Getting tasks off your critical path, reusability of documents (effort), possible quick wins, manageable process (metrics) and quality improvement are some of the advantages of structured testing that make it possible to get project leaders to accept the extra workload they will have to take on.

Although following the new methodology is compulsory for all merger projects, test consultants have to sell it to the project leaders. As long as they are not convinced of the added value, they will follow a look alike way of working that will slow them down rather than helping them. This selling requires the necessary skills and patience. A lot of techniques used to sell physical products were applied.

4.4 *Outline of the new methodology in the individual projects*

All the following activities are carried out by project members with the support of a test consultant. The project leader is responsible for all the test activities.

4.4.1 *Start of testing activities: Test plan*

As soon as possible test consultants invite project leaders to write the Test Plan(s) to get them to THINK ABOUT THINGS (Who's going to do what, where, when and how?). In the test plan the project specific approach and priorities for testing activities are defined. The project leader has to decide how he will spread the available resources over the different test activities, setting priorities according to risk and added value analysis.

4.4.2 *Create a Test Requirement Hierarchy (TRH) and assign Business and IT priorities to it*

On the TRH, which is the functional breakdown of the system under test, business and IT assign a priority or risk factor to each requirement. These priorities are used to define the detail in which requirements will be worked out into test designs, test cases and procedures.

A good design and scoring of the TRH is essential to achieve good test results because it is used as a base for most of the following testing activities. But sometimes this is easier said than done because business commitment and knowledge are not always available, especially in an organisation with a recently changed structure.

4.4.3 *Writing test designs, test cases and test procedures*

According to the priorities on the TRH, test designs, test cases and test procedures are written for the concerning requirements. The level of detail of these deliverables also depends on the type of project, the availability and stability of test data and the knowledge of the testers.

4.4.4 *Test execution*

Project leaders are responsible for test execution and management of test results and defects. They are supported by test consultants for tasks that go beyond their own project, especially during end-to-end testing.

4.4.5 *Defect tracking and reporting*

Follow up of defects and reporting test results within each project. Different tools are available to support project leaders with this.

Consolidation of test results, reporting and escalation of problems for items exceeding project limits are tasks of the test consultants.

4.4.6 *Evaluation*

To be able to improve the test process evaluation of the project is necessary. Not only test activities are scrutinised here, also test support, co-ordination and business participation are looked at.

Test consultants consolidate the results of the different projects, discuss them and determine adaptive actions.

4.5 *Organisation*

4.5.1 *Responsibilities must be known and taken*

End responsibility of the testing activities stays with project leaders who define commitments to be taken in the test plans. Test consultants give training and advice, they organise attunement meetings during which the evolution of test activities is discussed and deliverables are approved (after co-reading) and they bundle the testware of the different programs to be used as examples for future projects.

4.5.2 *Co-ordination must be set up*

- Test consultants

They support the co-ordination of end-to-end tests within their program, co-ordinate end-to-end tests that involve projects from different programs, they consolidate questions and solutions to build a knowledge base that can be used in current and in future projects and they check project interfaces and alignment of projects with release calendars.

- Environment support

A project was started to set up and manage the test environments. Due to the merger this includes a lot of extra effort. When this project finishes, the activities will be continued within the new (merged) organisation.

- Global Test co-ordination

A global test co-ordination team was set up to consolidate test results of all the programs, support daily and weekly escalation of problems and set up the Test forum to get everyone involved together on a weekly basis. They also co-ordinate the overall planning during end-to-end testing and assisting in the setting up and reservation of test data.

5 Lessons Learned

5.1 *Setting up the methodology*

- Keep templates generic, as they should be applicable to different types of projects. But beware, templates that are too generic tend to transform easily into unrecognisable forms that start to live a life of their own.
- Therefore it is important to use the original template every time you start a new “run”.

5.2 *Selling the methodology*

- When working with people that have different testing backgrounds, special attention must be paid to terminology. The same word may mean something completely different to two people coming from different companies;
- Sell the concept of structured testing first, next discuss the different activities as the projects make progress rather than trying to teach them everything in one go;
- Let test consultants be pragmatic, don't let them sell pragmatism.

5.3 *Test process improvement*

- Time and resources must be available to manage the methodology, otherwise it will start to live a life of its own, being changed continuously by different people, without keeping track of the changes;
- Manage your change process and use correct information and interpretation to define changes;
- Don't forget co-ordination of the co-ordination: steering and alignment of test consultants is necessary;
- TPI and test follow-up should be heard in mind from the beginning: you need to know what figures you will need at the end of your life cycle before you start producing templates.

5.4 *Test execution management*

- When time pressure increases, reduce your scope by setting new priorities rather than starting to rush things. To be able to do this, structured and well documented testing with the necessary metrics is required.

Biography

Lieven Schouwaerts is an engineer who started his career in the early 90's as a teacher at a technical university, teaching informatics, mechanics, electricity etc. After one year he started structuring logistic processes and implementing quality assurance systems for international companies. In the beginning of 1997 he joined GiTek nv where he has participated in projects as test-analyst, test co-ordinator and is currently test consultant for the implementation of structured testing at a large Belgian Bank. He is also responsible for the in-house support of Gitek's test engineers and consultants.



How to implement structured testing in narrow time boxed projects:

Rushing from chaos to structure?

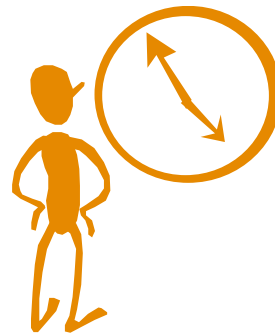
Lieven Schouwaerts

Gitek nv - *interaction through software*
St. Pietersvliet 3, B-2000, Antwerpen, Belgium
Tel: +32 3 231 12 90 - Fax: +32 3 226 10 83
E-mail: ls@gitek .be



Agenda

- Introduction
- Possible approaches
- Extra activities: going and growing concern
- How to bring this into practice?
- Lessons Learned



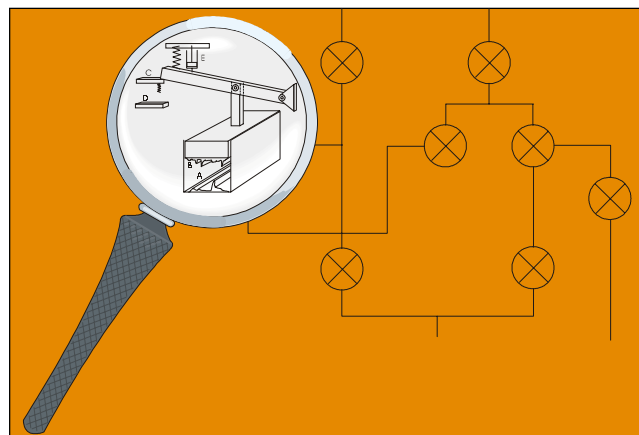


Introduction

- Background
- Rushing from chaos to structure?

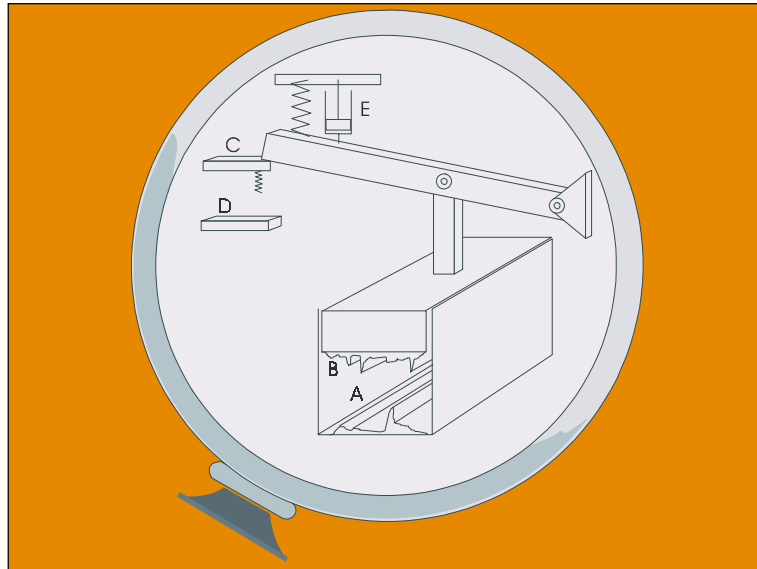


Possible approaches





Possible approaches



Possible approaches

- We didn't choose to
 - Structure one part of the process
 - Structure everything a bit



Possible approaches

- We didn't choose to
 - Structure only part of the process
 - Structure everything a bit
- We chose to set priorities according to risks, objectives and current maturity level



Extra activities: Going and growing concern

- BAU activities
- Y2K testing
- Euro follow-up
- Quality Assurance

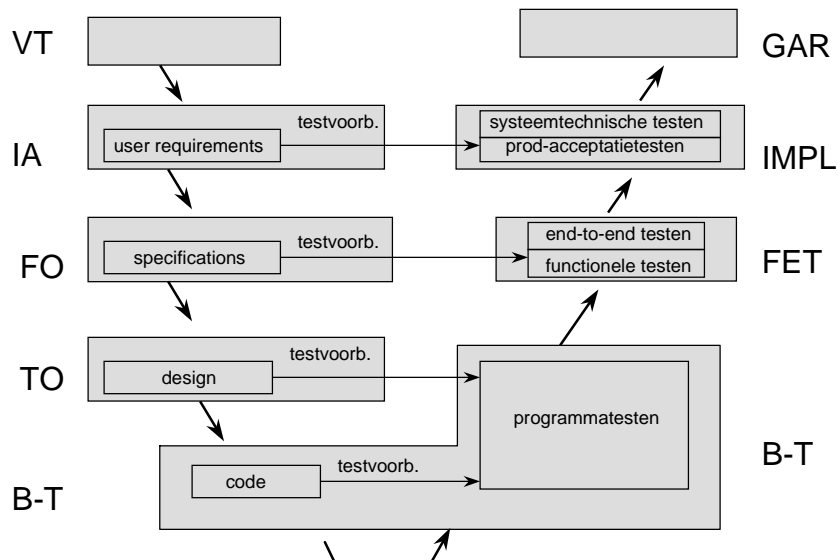


How to bring this into practice?

- Define priorities for the general approach
 - Changing the existing waterfall model into V-model (essential)



How to bring this into practice?





How to bring this into practice?

- Define priorities for the general approach
 - Changing the existing waterfall model into V-model
 - **Focusing on Functional and End-to-end Testing**
 - Support by test consultants
 - Merger projects



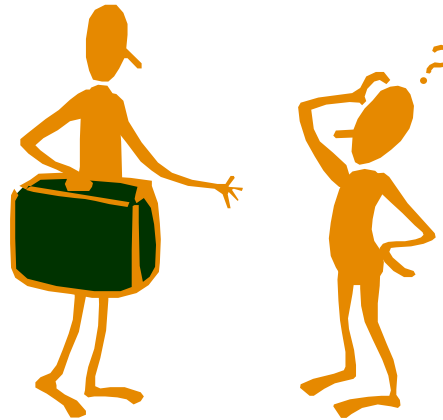
How to bring this into practice?

- Set up methodology
 - Test Plan
 - Test Requirement Hierarchy (TRH)
 - Test Design
 - Test Cases
 - Test Procedures
 - Test Execution
 - Defect Tracking
 - Test Report



How to bring this into practice?

- Sell methodology



How to bring this into practice?

- Work out activities in the individual projects
 - Start of testing activities: Testplan
 - Create a Test Requirement Hierarchy
 - Score TRH by Business and IT
 - Writing test designs, test cases and test procedures
 - Test execution
 - Defect tracking and reporting
 - Evaluation



How to bring this into practice?

- Organisation
 - Responsibilities must be known and taken
 - Co-ordination must be set up
 - Test consultants
 - Environment support
 - Global Test co-ordination



Lessons Learned

- Setting up the methodology
 - Keep templates generic



Lessons Learned

- Setting up the methodology
- Selling the methodology
 - Terminology
 - Sell the concept first, next discuss activities
 - Consultants can be pragmatic, not sell pragmatism



Lessons Learned

- Setting up the methodology
- Selling the methodology
- Test process improvement
 - Time and resources to manage the methodology
 - Manage your change process
 - Don't forget co-ordination of the co-ordination
 - TPI and test follow-up



Lessons Learned

- Setting up the methodology
- Selling the methodology
- Test process improvement
- Test execution management
 - When time pressure increases



Risk Based Test Strategy

Rob Baarda

Tim Koomen

IQUIP Informatica B.V.

P.O. Box 263, 1110 AG Diemen, The Netherlands

Tel: +31 20 660 6600

Fax: +31 20 695 3298

E-mail: baardaro@iquip.nl

koomenti@iquip.nl

1. Introduction

In recent years we have seen splendid opportunities (mostly Y2K projects) for the further development and implementation of test strategies. A test strategy is, in short, the choice which aspects of a (sub)system will be tested with what testing depth. We were able to combine theory on business risks with the existing theory of test strategy as described in TMap[®] (see references). This led to a straight line of thinking from business risks into detailed testing depth per function.

One of the opportunities to implement ideas on test strategy came out of the Y2K projects. In Y2K projects large numbers of programs had to be tested. Due to limited resources not all programs could be tested. How to make the choice which programs to test and which not? We used the test strategy as an approach to make these choices.

Some organisations started in parallel to change from the intuitive way of making a test strategy into a more rational one based on business risks.

In this paper we first describe the theory from business risk unto test techniques, followed by two concise case-stories of the previous described situations.

We will make clear that the described manner to define a test strategy has the advantage of:

- better test coverage on the right spot;
- improved communication between the tester and other parties concerned.

2. Test strategy and risks

The development of a test strategy is a means of communication with the customer commissioning the test on such matters as the organisation of testing and the strategic choices that go with it. The test strategy indicates how testing is to be carried out. In order to make the best possible use of resources and time, it is decided on which parts and aspects of the system the emphasis should fall. The test strategy forms an important basis for a structured approach to testing and makes a major contribution to a manageable test process.

The customer who commissions the test will expect specific qualities of the system when in production, and wants to know whether the released system will meet these requirements. If the system qualitatively does not meet the requirements or only to a limited extent, this implies high damage for the organisation, for instance since high rework costs will be needed or clients/users will be unsatisfied. Therefore, this situation forms a risk for the organisation. 'Risk' in this paper is defined as:

A risk is the chance of an error¹ occurring (chance of failure) related to the damage expected when this error does occur

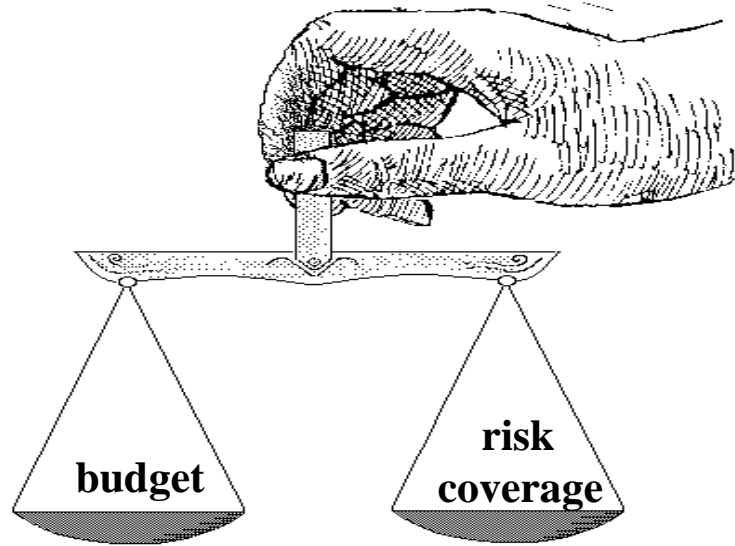
Testing covers such risks by giving insight into the extent to which the system meets the quality demands. When quality turns out to be insufficient timely measures can be taken, e.g. rework by developers. If the shipping of the system implies many risks for the organisation, better testing is obvious as a solution. And the reverse also holds:

No risk, no test

Although in the above we refer to quality and risks in a general sense, there may be large differences depending on the situation. It is of great importance to discuss this with the customer, and to translate the customer's wishes in this respect into the way testing will be performed. Thus, the test strategy is directed towards finding the optimal balance between the test effort to be exerted and the coverage required for the risks. To this purpose the risks are specified up to the level of quality characteristics and separate subsystems. In doing so it becomes possible to find a suitable test coverage for the assessed risks. Here a higher test coverage usually results in more test effort. In order to reach at the variation in test coverage needed, the use of more than one test specification technique (test design technique), each offering a specified test coverage, is crucial.

An analogy with insurance industry may clarify this matter a bit more. A person wants to cover a relevant risk and takes an insurance with a coverage fitting this risk as best as possible. This insurance takes a certain premium. If the person wants to pay less, an insurance with a lower coverage is bought. The consequence is that there will be no payment if the uncovered risk occurs. On the other hand, if coverage was too large, then too much premium is paid, since a situation has been insured which is unlikely to occur for this person.

¹ The terms error, defect and failure are not used as exactly as IEEE advocates. In this paper error = fault or mistake; failure = the result or manifestation of one or more errors.



The balance between budget and risk coverage

3. Risk Assessment

Test strategy is based on risk assessment. This means assessing the damage of the consequences of failures, both undetected prior to operation and occurring during operation.

Risk assessment takes place on the basis of quality characteristics and subsystems. For instance, if the system is insufficiently user-friendly, what will be the negative consequences. And what will be the damage when the salary calculation module in a payroll system does not work correctly.

In order to be able to perform this assessment well, the separate aspects of a risk are considered:

$$\text{Risk} = \text{chance of failure} \times \text{damage},$$

where chance of failure is related to aspects including frequency of use and the chance of an error occurring.

These aspects are listed below:

- Frequency of use

In a function which is used dozens of times each day the chance of an error demonstrating itself is much bigger than with a function used once a year.

- Chance of error

For the assessment of the chance of errors the following list can be helpful. It presents the locations where errors tend to cluster. It is partly based on H. Schaefer, 1996 (*Surviving under time and budget pressure*, in: Conference Proceeding EuroSTAR1996, Amsterdam, the Netherlands):

- ◇ Complex functions;
- ◇ Completely new functions;
- ◇ (Especially frequently) adjusted functions;
- ◇ Functions for which certain tools or techniques were employed for the first time;

- ◇ Functions which were transferred from one developer to another during development;
- ◇ Functions that were realised under extreme time pressure;
- ◇ Functions which had to be optimised more frequently than on average;
- ◇ Functions in which many errors were found earlier (e.g. in previous releases or during earlier reviews);
- ◇ Functions with many interfaces;
- ◇ Inexperienced developers;
- ◇ Insufficient involvement of users;
- ◇ Insufficient quality assurance during development;
- ◇ Insufficient quality of low-level tests;
- ◇ New development tools and development environment;
- ◇ Large development teams;
- ◇ Development teams with sub-optimal communication (e.g. owing to geographical spread or personal causes);

- Damage

If and when the error manifests itself, what will be the damage for the organisation. Aspects are costs of repair (both of the system and of the consequences), forgone income and loss of clients or of confidence. Usually the damage increases if the error has its impact on other functions or systems. In the case of errors occurring in batch processes there may be a possibility to prevent them from hampering users, so that the eventual damage will be smaller than with similar on-line processes. Of course, this only holds if errors are detected on time.

Because of the complexity of the matter, it is impossible to assess risks with complete objectivity and in detail: it is a global assessment. It is therefore important for the risk assessment not to be carried out by the test manager alone. A large number of people involved in the scheme should contribute: customer, users, development team, accountants, IT auditors and so on. This not only increases the quality of the strategy, but it also has the advantage that the different parties are more aware of the risks and the extent to which testing contributes to making these risks manageable in a better way.

The developer of the test strategy should realise that 'users' are the best people to assess the damage and the frequency of use when valuing the risks (end-users, system managers and application managers, line management), whereas project team members are best to assess the chance of error (project managers, designers, programmers, project quality staff, test manager).

The focus in risk assessment is on product risks, or, in other words, what is the risk for the organisation if the product does not demonstrate the expected quality. In addition to this, there are also (test) project risks. If the system must be in production on January 1st, if functional specifications are produced too late, if no experienced testers are available, or if the test infrastructure is not ready on time, then we speak of (test) project risks. These are not taken into account in determining the test strategy; they do play a role in the test plan.

In developing a test strategy the aim is to see to it that the test will be organized in such a way that with a certain extent of reliability

- the most important problems will be found;
- the problems will be found in an early stage;
- the problems that require the most rework time will be found first;
- efficient use is made of resources;
- and eventually an accurate quality advice can be given.

This can be summarised as:

Test strategy aims at finding the **most important** errors as **early** as possible against the **lowest costs**

In practice, the development of a test strategy is often planned to coincide with preparing the budget, for example with the help of test point analysis. The advantage is that the consequences of the adopted strategy are immediately translated into time required for testing, and consequently the cost of testing, which makes the strategic choices manageable. If the time available for testing is more or less fixed, it is also possible to use test strategy combined with test point analysis to determine what is achievable within the time limits. It is probably even more important to make it clear at this time which parts cannot be tested, or cannot be fully tested, and what risks will therefore be incurred.

4. Quality Characteristics

The quality characteristics we distinguish can be divided into dynamic and static quality characteristics. The dynamic quality characteristics deal with features of the information system in use; examples are security, usability, continuity, traceability, functionality, userfriendliness, suitability, efficiency, performance. The static are concerned with intrinsic characteristics of the information system and the documentation, as considered from the standpoint of developers and future system managers. Examples are manageability, maintainability, connectivity, reusability, portability, testability.

5. Steps

The development of a test strategy is not something that can be done purely methodically or formally. The below steps are aids and indicators. Experience and skills of the performer of this activity in the area of testing is a major success factor for a sound test strategy.

One should also realise that test strategies arise as a result of iterative processes and in connection with other activities for a test plan. If the first test strategy produces an amount of test effort needed or a certain time schedule which is unacceptable for the customer, the strategy should be adjusted. The lack of test skills or suitable infrastructure can also result in adjustments of the test strategy.

The stepwise defining of the test strategy can be used for any test level and also for an overall strategy (master test plan), including and co-ordinating all test levels and even inspections. The steps differ for both situations.

5.1 Strategy in Master Test Planning

The steps to be taken for a test strategy are:

- Decide on the quality characteristics;
- Determine relative importance of quality characteristics;
- Attribute quality characteristics to test levels.

5.1.1 Step 1: Selection of Quality Characteristics

In close liaison with the customer and other parties involved a selection of quality characteristics is made on which the tests must focus. In doing so one should take risks for the business into account as well as aspects including system requirements, business objectives concerning the information system, directions and standards set by the computer centre. These quality characteristics are also used for reporting to the customer during test execution and completion.

Some characteristics are difficult to test. There may be a wish for a system to be user-friendly and flexible, for instance, but these wishes turn out not to have been translated into measurable requirements. That is why a substantial part of the effort here is devoted to formulating the relevant quality demands as measurably and unambiguously as possible. It is also the case that some quality characteristics demand relatively much effort in testing. Since it is not useful to offer possibilities which cannot be fulfilled, it should be determined beforehand what will be the estimated effort needed for a decision made.

For non-IT people our quality characteristics may be hard to handle. It helps when we translate them to the conceptual environment of our conversational partners. This can be done by finding illustrative examples of problems or errors that may occur in production and the damage that would be caused by this. This is one of the most difficult aspects of the formulation of a test strategy.

5.1.2 Step 2: Relative Importance of Quality Characteristics

On the basis of the results from Step 1 the importance of the selected quality characteristics is determined in relation to one another. This is done in the Matrix of Weights (see below), by weighing the relative risks per quality characteristic. Here the relative importance is indicated (in percentages). Note that it is not of importance to have exact percentages: the objective is to arrive at a general picture of the relative importance of the various quality characteristics. The filling in of the matrix helps evaluating the risks.

The customer should be forced to make choices. Therefore, as a directive we ask for a percentage of 5 as the minimum. The sum of all percentages should not exceed 100. An example of a Matrix of Weights is given below:

| Quality characteristic | Relative importance |
|------------------------|---------------------|
| Manageability | 5 |
| Security | 5 |
| Usability | - |
| Connectivity | - |
| Continuity | 10 |
| Traceability | - |
| Flexibility | - |
| Functionality | 50 |
| Userfriendliness | 10 |

| | |
|-----------------|------|
| Reusability | - |
| Infrastructure | - |
| Suitability | 10 |
| Maintainability | 5 |
| Performance | 5 |
| Portability | - |
| Testability | - |
| Efficiency | - |
| Total | 100% |

The Matrix of Weights

The high percentage for functionality in this matrix may strike the reader. This is in conformance with practical experience: generally 50% of the importance or more is attributed to this characteristic. The reason for this is that risks usually are larger for incorrect performing systems (Functionality) than for slow systems (Performance) or awkward systems (Userfriendliness).

5.1.3 Step 3: Quality Characteristics Attributed to Test Levels

With the aim of spending the total test effort as efficiently as possible, during test strategy development it is decided with which test level or combination of test levels the various selected quality characteristics will be tested. Also inspections may fall under the scope of the master test plan and under the test strategy. In the remaining sections when 'test' is used, inspections are also included.

In this way the various test levels within a project are brought into balance. It is obvious that the different responsibilities and authorities remain intact.

A +-sign in a matrix (for an example, see matrix below) indicates whether the test strategy takes a quality characteristic into account. '++' or '+++' indicate that relatively much attention is to be paid to the quality characteristic for the specified test level. It is obvious that one quality characteristic can be in effect for more than one test level, but depth will often vary. If structured test specification techniques are used, the acceptance test, for example, may use results of previous tests levels, on the basis of which it may be decided to test with less depth.

| | Insp RQMS | Insp Specs | Insp Design | PT | IT | ST | FAT | PAT | Relative importance |
|---------------|-----------|------------|-------------|----|----|----|-----|-----|---------------------|
| Manageability | + | + | ++ | | | | | + | 5 |
| Security | + | + | + | | | | + | + | 5 |
| Usability | | | | | | | | | - |
| Connectivity | | | | | | | | | - |
| Continuity | + | | + | | | | | ++ | 10 |

| | | | | | | | | | |
|------------------|----|----|---|---|---|-----|----|---|------|
| Traceability | | | | | | | | | - |
| Flexibility | | | | | | | | | - |
| Functionality | ++ | ++ | | + | + | +++ | ++ | | 50 |
| Userfriendliness | | ++ | | | | | ++ | | 10 |
| Reusability | | | | | | | | | - |
| Infrastructure | | | | | | | | | - |
| Suitability | + | ++ | | | | | ++ | | 10 |
| Maintainability | | + | | | | + | | | 5 |
| Performance | | | + | | | | + | + | 5 |
| Portability | | | | | | | | | - |
| Testability | | | | | | | | | - |
| Efficiency | | | | | | | | | - |
| | | | | | | | | | 100% |

Example of a Test Strategy for Test Levels

Legenda:

| | |
|-------------|--|
| Insp RQMS | Inspection of Requirements |
| Insp Specs | Inspection of Functional Specification |
| Insp Design | Inspection of Technical Design |
| PT | Program Test |
| IT | Integration Test |
| ST | System Test |
| FAT | Functional Acceptance Test |
| PAT | Production Acceptance Test |

5.2 Strategy for a Test Level

The steps to be taken for a test strategy for a specific test level are:

1. Decide on the quality characteristics;
2. Determine relative importance of quality characteristics;
3. Divide the system into subsystems;
4. Determine relative importance of subsystems;
5. Specify test importance per subsystem and quality characteristic;
6. Establish test techniques to be used.

The strategy determination for a specific test level naturally has the master test plan strategy as a precondition and a starting point. If a master test plan, including a test strategy, is there, step 1 can be omitted and step 2 will be an easy and fast performed activity. Nevertheless, all steps are worked out below.

5.2.1 Step 1: Decide on Quality Characteristics

In collaboration with the customer and perhaps other parties concerned the quality characteristics are determined on which the test will focus, in relation to business risks. During the test and in the completion phase, results are reported on the basis of these quality characteristics.

5.2.2 Step 2: Determine Relative Importance of Quality Characteristics

Based on the results of step 1 the relative importance of the selected quality characteristics is determined. Determination of the importance takes place by weighing the risks per quality characteristic. This is shown in a Matrix of Weights by a percentage in the column Relative importance. In order to force the customer to make choices, a percentage of 5 is the minimum.

An example of a matrix for a functional acceptance test is given below:

| Quality characteristic | Relative importance |
|------------------------|---------------------|
| Security | 5 |
| Functionality | 60 |
| Userfriendliness | 10 |
| Performance | 5 |
| Suitability | 20 |
| Total | 100% |

The Matrix of Weights for a Functional Acceptance Test (Example)

5.2.3 Step 3: Divide System into Subsystems

During this step and the following steps the test strategy is refined more and more. This implies that the quality characteristics and their relative importance as indicated in the Matrix of Weights are to be broken down for the combination of test specification technique and subsystem, later even for test specification technique and test unit.

The information system is divided into subsystems. The reason for this is that the same quality demands do not have to be valid for each subsystem. Moreover, the various subsystems may have different risks for the organisation. In principle the division is the same as given in the design documentation. If we deviate from this one, we must clearly indicate the motivation for this. Examples of alternative divisions are on the basis of extent of risk or on the basis of order of release by the developer. If a conversion module is there, this is to be treated as a separate subsystem. Often the subsystem 'Total system' is distinguished. This serves the purpose of indicating that some quality characteristics can be evaluated effectively only with the help of an integral test, testing the coherence of the various subsystems.

In a later stage the various subsystems are further divided into independent test units. E.g. in a logistics system the subsystem Sales may be divided into the test units Quotations (all functions regarding quotations) and Orders.

5.2.4 Step 4: Determine Relative Importance of Subsystems

On the basis of the result of the previous step the relative importance (in percentages) of the subsystems should be indicated in the Matrix of Weight. This is done by weighing the risks per subsystem. It is not a matter of exact percentages; rather it is a matter of getting a general image of the importance of the subsystems as seen through the eyes of the customer and other parties concerned. This step helps in asking people to form an opinion of this.

The relative importance is determined of each subsystem within the information system. In the Matrix of Weights this is indicated with a percentage in the column Relative importance.

An example of a Matrix of Weight for a functional acceptance test (based on the Strategy Matrix for test levels in the master test plan, shown above) is given here:

| | Relative importance |
|--------------|---------------------|
| Subsystem 1 | 30 |
| Subsystem 2 | 15 |
| Subsystem 3 | 20 |
| Conversion | 15 |
| System | 20 |
| Total | 100% |

Relative Importance of Subsystems for a Functional Acceptance Test (Example)

5.2.5 Step 5: Specify Test Importance per Subsystem and Quality Characteristic

Finally a refinement is made by assessing the importance of the combination quality characteristic - subsystem. E.g., a refinement may be that userfriendliness is important (relative importance of 10), but this holds predominantly for subsystem 1 and not at all for subsystem 3. Again it is emphasised that test strategy determination is not a mathematical affair: it is meant to get an image of the relative test importance of the various subsystems and quality characteristics. This is also the reason why we choose +, ++ and +++ as notation symbols, rather than opting for the pseudo-certainty of a mathematical formula. An example of this is the following: suppose both userfriendliness and a specific batch subsystem are very important, a mathematical formula would probably result in large test effort to be spent on the userfriendliness of the batch procedure. The Matrix of Weight may look like this:

| | Subsystem 1 | Subsystem 2 | Subsystem 3 | Con-version | Total system | Relative importance |
|---------------------|-------------|-------------|-------------|-------------|--------------|---------------------|
| Security | + | + | | | | 5 |
| Usability | | | | | | - |
| Continuity | | | | | | - |
| Traceability | | | | | | - |
| Functionality | ++ | + | + | ++ | + | 60 |
| Userfriendliness | ++ | + | | | | 10 |
| Performance | + | | + | | | 5 |
| Suitability | + | + | + | | ++ | 20 |
| Efficiency | | | | | | - |
| Relative importance | 30 | 15 | 20 | 15 | 20 | 100% |

Relative Importance of Subsystem x Quality Characteristic (Example)

5.2.6 Step 6: Establish Test Techniques to be Used

The final step in test strategy involves the selection of the test specification techniques that will be used to test the combination of the selected quality characteristics and subsystems. A high importance implies the use of techniques with a high coverage or the use of more techniques, a low importance implies the use of techniques with a lower coverage or the use of fewer techniques.

In choosing the techniques one should also take into account various other factors, a number of which are listed below.

- Quality characteristic to be tested
- A technique is fit for testing one or more quality characteristics. Some quality characteristics can best be tested with one (set of) techniques, others with another one.
- Area of application

Some techniques are specifically suitable for testing the interaction (screens, reports, on-line) between the system and the users, others are better in testing the processing of systems (batch processes). There is a relation with the type of error to be found with a technique, e.g. false input checks, incorrect processing or errors of integration.

- Availability of test basis

Each techniques starts from a certain test basis. This may be the functional specification, the technical design, program code or descriptions of the end-user organisation. The exact form of the test basis is also relevant to the choice of a technique, e.g. decision tables, pseudo-code, structured language or unstructured prose.

- Extent of formality

Informal test specification techniques offer more freedom for the tester in making the test cases than do formal techniques.

- Use of resources

The application of a techniques requires a specific amount of resources, in terms of man capacity as well as machine capacity. The use of resources has a direct relation with costs.

- Required knowledge and skills

Not each tester is equipped for each technique. For the useful application of some techniques much business knowledge is needed. For other techniques more analytic talent is required. Therefore, the knowledge and skills of the test staff also influences the choice for techniques.

For practical reasons one should attempt to cover all selected quality characteristics with a minimal set of test specification techniques.

The selection of the test specification techniques should be done in an early stage of the test process, for then the test team can take the appropriate actions in training for techniques and the necessary checklists can be made or adjusted for the specific situation.

As a result of this step the techniques that will be used per subsystem are defined. Optionally, especially with large test projects, this last step in the test strategy is performed slightly later in the process, namely during the preparation phase. As a part of this the priority order of the tests to be performed is determined. The aim of this is to have the most important tests take place as early as possible.

6. Case-stories

The case-stories are an illustration of some parts of the presented theory. In both situations the organisations have chosen for the quality characteristic functionality. The test level is in both cases system (regression) testing.

The case-stories are simplified and not extensive.

6.1 Y2K case-story

Situation

A large financial institute, with more than 100 (sub)systems, wants the systems to be Y2K-compliant. One of the tests to be passed is the system regression test after the Y2K-adaptation. There is a limitation on the number of systems that can be tested due to lack of infrastructure.

Which systems have to be tested?

Detailed risk analysis

The answer was found in making a prioritisation of the (sub)systems based on a risk analysis. The risk analysis is based on the formula:

$$\text{Risk} = \text{Chance of failure} * \text{Damage}$$

The damage component of the (sub)system was rated by the impact of a failure for the business process using a scale from 1 (small impact) to 5 (large impact).

In this project the definition of the chance of failure was based on attributes which are relevant in Y2K:

$$\text{Chance of failure} = [\text{Size of system}] * [\text{Complexity and size of Y2K-adaptation}]$$

The size of the system could not easily be measured but was estimated (between 1 and 5) based on data like:

- lines of code
- number of function points
- number of files
- number of interfaces

The complexity and size of the Y2K-adaptation was judged between 1 and 5 using data as:

- number of (expected) changes, related to the number of dates
- quality of the system documentation
- available knowledge about the (sub)system

If the number of changes was zero, the Y2K-adaptation size was zero. This leads to a zero chance of failure and to zero risk, so regression testing was not necessary.

Result

It was possible to rate each (sub)system with a risk number. Of course the (sub)systems with a high risk number were regression tested in the regression test infrastructure, and the (sub)systems with a low number were not. Intermediate systems were only tested if infrastructure resources were left.

All participants, management, users, testers, felt comfortable because all choices were made clear for everybody.

The risk based thinking helped to make an objective choice which systems had to be tested.

6.2 Testing depth case-story

Situation

An organisation introduced test tools and wanted all functional tests to be automated for regression test purposes. The organisation used a certain test specification technique based on decision tables for the design of the test cases. This technique can be applied with a variable testing depth. The larger the testing depth, the larger the coverage of the test cases, but also the higher the test effort. A first planning based on the maximum testing depth showed that the effort to test was larger than the available budget.

Choices had to be made: which testing depth to apply on which function?

Detailed risk analysis

It was decided to break down the system into functions. For each function the chance of failure was estimated by the programming staff, the expected damage was decided by the user organisation. Both scales were 1 to 3, so the calculated risk varied from 1 to 9.

To cover risk number 9, it was decided to give a full testing depth; e.g. if there were 8 test cases possible, they were all realised.

Functions with risk number 6 were tested with 4 out of 8 test cases, and functions with lower risk numbers got only one or two test cases out of the 8.

Result

The regression test set was put together in the described way, sometimes after long discussions between users and programmers. In production there were no high damage failures, so it looks like that there were no test cases missing.

Management, testers, users and programming staff had the idea that the available budget was used in an optimal way.

7. Conclusion

The testing of information systems can be based on the business risks which the organisation will experience in using these information systems if the system is not tested. In practice, test managers often take the steps to come from risks to test coverage in an intuitive manner. In this paper, the steps needed for the definition of a test strategy are made explicit. The result of such a test strategy is better insight for all parties involved and a sound basis for negotiating testing depth.

Good risk assessment is a part of these steps. It is essential to realise that this explicit way of looking at risks cannot be done by a test manager or tester alone. It is necessary to ascertain for the involvement of users and managers of the client organisation, of auditors, and of project people such as developers, testers, QA staff and project managers. In practice, the discussion of risks and related testing strategies in this way proves to be a real eye-opener for all parties concerned. It also enables negotiation of testing depth by having the customer decide which elements should be tested how thoroughly.

The stepwise definition of the test strategy can be used for any test level (e.g., system test, acceptance test) and also for an overall strategy (master) test plan, including and co-ordinating all test levels and inspections/reviews.

8. References

In English with a short description of test strategy
Pol, Martin and Veenendaal, Erik van, Structured Testing of Informations Systems (1998), Kluwer Deventer The Netherlands, ISBN 90-267-2910-3,

In Dutch with an extensive description of test strategy
Pol, Martin, Teunissen, Ruud and Veenendaal, Erik van, Testen volgens TMap (1995), Tutein Nolthenius, s' Hertogenbosch The Netherlands, ISBN 90-72194-33-0;

In December 1999 this book will be available in a new version describing the above theory. ISBN 90-72194-58-6.

Risk Based Test Strategy

Rob Baarda baardaro@iquip.nl
Tim Koomen koomenti@iquip.nl
IQUIP Informatica B.V.
The Netherlands

IQUIP
98 nr Be 1

Agenda

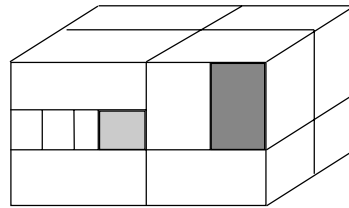
- Test strategy
- Risk analysis
- Procedure
- How to apply
- Case
- Conclusion

IQUIP
98 nr Be 2

Test Strategy

Aim: To detect the most important defects as early as possible at the lowest costs!

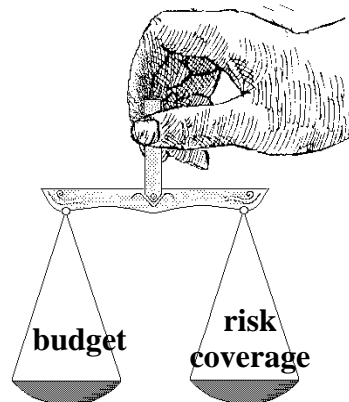
What are the most important defects?



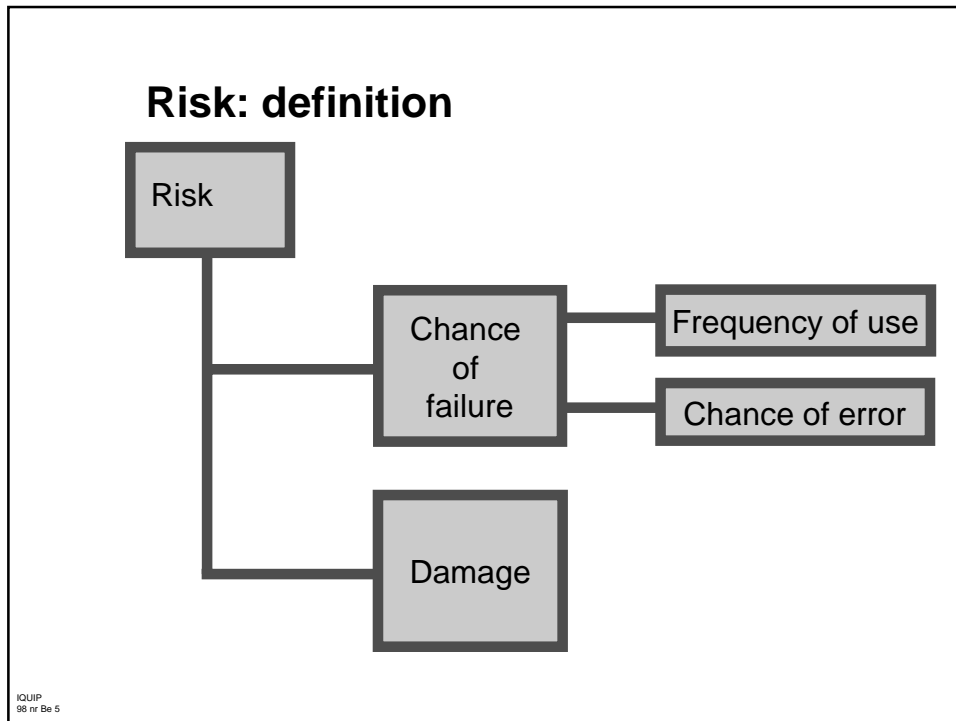
IQUIP
98 nr Be 3

Risk thinking: an introduction

- Business reasons
- No risk, no test
- Risk analysis approach



IQUIP
98 nr Be 4



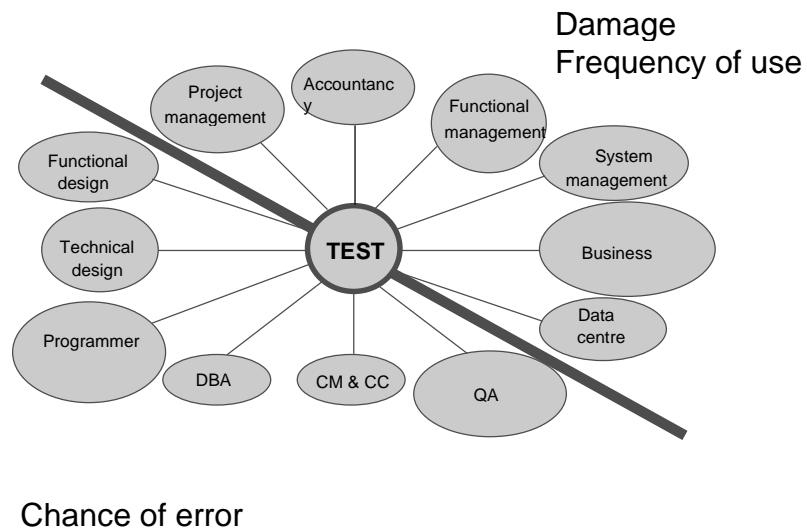
- ### Risk definition details
- Damage
 - financial, loss of faith of customers, damage to corporate identity
 - impact other functions and/or systems
 - detection and repair time
 - Chance of error
 - global = size * complexity
 - detailed = knowledge of development quality
- IQUIP
98 nr Be 6

Risk analysis practical

- Applicable on the level of:
 - system
 - subsystem
 - individual function (e.g. interest calculation)
- Risk analysis should lead to a limited number of classes of (more or less) equal risks
- Applicable on quality characteristics, what is the damage and the chance that it will happen:
 - functional defects
 - low performance
 - low maintainability
 - ...

IQUIP
98 nr Be 7

Parties involved



IQUIP
98 nr Be 8

Steps

Master Test Plan

- selection of quality characteristics
- relative importance of quality characteristics based on risks
- quality characteristics attributed to the test levels

For one test level

- determine (relative importance) of system parts based on risks
- specify test importance per system part and quality characteristic
- choose test techniques (TT) and testing depth

IQUIP
98 nr Be 9

Procedure visualised

| Quality characteristics | Relative importance % | <u>System part/Risk class</u> | | | |
|-------------------------|--------------------------|-------------------------------|-----------|----------|----------|
| | | 1 | 2 | 3 | System |
| Security | 5 | | | | + TT2 |
| Functionality | 75 | ++ TT1 | ++ TT1 | + TT2 | + TT2 |
| Usability | 20 | + TT3 | + TT3 | + TT3 | |
| ... | 0 | | | | |
| | 100 % | 40 | 25 | 15 | 20 |

IQUIP
98 nr Be 10

How to apply

A documented Risk Based Test Strategy helps:

- to be complete in assessing the risks
- discussions between parties
- to get commitment of the customer
- later to:
 - have documented proof if needed
 - make a switch in test manager position easier
 - do a re-planning
 - use in the maintenance situation

IQUIP
98 nr Be 11

Case-story Y2K

- Situation
 - large financial institute
 - > 100 (sub)systems going for Y2K-compliance
 - limited regression test capacity
- Which systems to choose?
- Approach: Risk based
- Risk = Chance of failure * Damage
 - chance of failure for Y2K adaptations [1-5]
 - damage = importance for business process [1-5]

IQUIP
98 nr Be 12

Case-story Y2K: example

| System | Chance of failure | Damage | Risk |
|--------------------|----------------------|--------|------|
| Debt mgt | 5 | 5 | 25 |
| Cash mgt | 3 | 2 | 6 |
| Stock transactions | 1 | 5 | 5 |

great risk (25): regression test

middle risk (6-7): regression test if is capacity left

IQUIP
98 nr Be 13

Case-story Y2K: results

- all parties involved in time
- with a good communication about the estimations
- optimal use of the regression test infrastructure
- most risky systems tested in time

IQUIP
98 nr Be 14

Case-story testing depth

- Situation
 - card issuing organization
 - building a regression test for more systems
 - limited budget
- Which functions to test thoroughly?
- Risk based!

IQUIP
98 nr Be 15

Case-story testing depth: example

- risk for each function = chance of failure * damage
- chance from 1 to 3 by programming staff
- damage from 1 to 3 by users
- possible risk numbers: 1, 2, 3, 4, 6, 9
- test technique based on decision tables
- risk number = 9: all combinations
- risk number = 4 till 6: 50% of combinations
- risk number = 1 till 3: some combinations

IQUIP
98 nr Be 16

Case-story testing depth: results

- good communication between users and programming staff (after some time)
- in production no high damage failures occurred afterwards
- a manageable test process for the test manager

IQUIP
98 nr Be 17

Conclusion

Risk Based Test Strategy

- applicable for different test levels
- supports users to make choices
- is a medium to have communications between users, programmers, test management
- makes the test process manageable

IQUIP
98 nr Be 18

Managing the Transition from ISO to High Maturity Levels of the CMM

powered by intellect.
driven by values.



Managing the Transition from ISO to High Maturity Levels of the CMM
Managing the Transition from ISO to High Maturity Levels of the CMM
Managing the Transition from ISO to High Maturity Levels of the CMM
Managing the Transition from ISO to High Maturity Levels of the CMM
Managing the Transition from ISO to High Maturity Levels of the CMM

ISO to CMM

K. Dinesh
Pankaj Jalote
M. R. Bhashyam
S. Raghavan

Infosys Technologies Ltd.
Electronics City
Bangalore – 561 229
Fax: +91-80-852-0352
Email: kdinesh@inf.com



ABSTRACT

In an effort to improve its processes, Infosys, a large ISO9000-certified software house, adopted the CMM framework. Within one year, Infosys was able to successfully transition from ISO to level 4 of the CMM. A key success factor in this achievement was that the transitioning was treated and managed like an aggressive project. This article describes some of the important aspects of managing this transition.

Keywords: Software process improvement, ISO9001, Capability maturity model, Process maturity, Project management.



INTRODUCTION

A process framework specifies some characteristics that the process must have in order to “qualify” to be a process of some maturity. The maturity of a process may be classified in some levels, and the number of levels in which a framework characterizes a process may be two or more. A framework only specifies the characteristics that processes at different levels should have and does not prescribe any process so that different processes may fulfil the requirements of the framework. By specifying characteristics of processes for different levels of maturity, frameworks also provide guidance regarding the improvements needed to move from one maturity level to the other.

There are many frameworks for software processes. Currently, the two most used and influential models are ISO9001[5] and the CMM [9]. ISO9001 is a general standard for providing service, which has been specifically interpreted for software in ISO9000-3 [6]. TickIT provides further guidelines on how ISO9001 is to be used by software organizations [7]. ISO9001 has 20 clauses which an organization must satisfy in order to qualify to be “ISO certified”. Within the “ISO certified” category, there are no distinctions (further improvement is generally handled through the auditing process). In other words, there are only two levels in the ISO9001 framework. The model is general and considers the working of the whole organization, not just of its software projects.



The CMM for software is a framework that focuses on processes for software development whose foundations were laid down in [4], and the framework itself is described completely in [9]. One of the objectives of the CMM is to distinguish mature processes from immature or ad-hoc processes. In the path to higher maturity, there are some well defined plateaus that are viewed as *maturity levels* by the CMM. The CMM specifies five maturity levels in this path – level 1 (which is the lowest) to level 5 (which is the highest). Each maturity level (except level 1) is characterized by some Key Process Areas (KPAs), which specify the areas in which the organization should focus if it wants its process to be at that maturity level. Of the 700 assessments that were done between 1992 and 1997 and whose assessment results are with the SEI, about 165 organizations were assessed at level 2, about 105 at level 3, about 16 at level 4, and about 4 at level 5 [12]. That is, of the 700, only about 20 organizations are at levels 4 or 5. The number of high maturity organizations is growing rapidly however.

Infosys is a large software house, currently employing over 3000 employees. It provides software services to customers from around the world, and executes over 200 projects each year. Its business has been growing at the rate of about 70% per year for the last five years. In its early growth stages itself Infosys



recognized that a rapid pace of growth cannot be managed without a set of properly defined processes for executing and controlling software projects. To achieve this, initially, it adopted the ISO 9001 TickIT model leading to ISO certification. After the ISO compliant systems were established, a need was felt for improving the processes further. It was felt that for further process improvement, ISO provided little guidance, and the CMM framework for process improvement was adopted. Having implemented ISO, the organization was somewhere between level 2 and 3 (implementing ISO generally implies that most of the Key Practice Areas (KPAs) for level 2 of CMM are generally satisfied and same portions of same KPAs at level 3 may also be satisfied [10]). A limited assessment by an external consultant had also placed the organization at level 2. Early in 1997, due to business requirements and other needs it was decided to move aggressively on the adoption of CMM. In less than a year after this decision, Infosys was successfully assessed to be at level 4 of the CMM. This article describes some of the key aspects and strategies in taking a project-management approach to managing the transition from ISO to CMM. It is generally believed that process improvement is a slow process which must be done gradually. Our experience suggests that “rapid process improvement” is possible with proper management.





SETTING THE GOAL

There are two approaches for implementing the CMM (or any other) framework. One is to do process improvement and enhancement based on needs and analysis, and then later go in for an assessment. In this approach, achieving a level is essentially a “side effect” of software process improvement initiatives. The other approach is to fully accept the CMM framework, set some target in terms of maturity level, and then strategize and plan accordingly.

If a project approach is to be followed, then as in any project, the goal or the desired end result must be extremely clear. The SEI survey also reports that having well understood software process improvement (SPI) goals is a key success factor [2]. Furthermore, everyone involved in the project must be fully committed to achieving the goals. As has been pointed out in [11], a shared vision is very important for the success of SPI initiatives. Though goal setting and obtaining commitment can be done with the first approach, they are considerably easier in the latter as the goal is very clear – achieving a maturity level. Furthermore, demonstrating that goals have been met or progress is being made towards achieving the goal is much harder in the former approach and can require a considerable period of time to collect enough data to “prove” the case. This exercise is much simpler in the latter – the fulfillment of the goal can be demonstrated through an assessment.

At Infosys we followed the latter approach. The lowest level Infosys could shoot for was level 3, but it was felt that this goal is not sufficiently ambitious and will add only marginal value. As going for level 4 added only two more KPAs it was felt that with some additional effort we should be able to achieve it. Also level 4 will provide the quantitative visibility & analysis that the organization desired. Due to these reasons, and some business reasons, the target of level 4 was set for the company. The senior management of the company, including the CEO in co-operation with the head of SEPG, set the target. Indeed, a goal like this



cannot be set by the SEPG itself and has to be set by the top management. This made (and would generally make) the senior management and the SEPG partners in the SPI initiative, providing the necessary buy-in by senior management, which is important for the success of any SPI initiative [1, 2, 3], and avoiding the potential problem of lack of senior management support [2, 3, 11].

Once the target level was set, the schedule was to be decided. As a rough KPA-wise gap analysis suggested that gaps are manageable, a schedule of one year was set. A short time span was given to suggest the “importance” of the initiative, which is also an important success factor [2, 3]. One year was also long enough to effectively complete one full cycle of process improvement, along with any corrections that might be needed in that cycle. Though moving up the maturity scale is generally considered as a slow process and experience of many is that it takes longer than expected [3], we thought that with a focused target, and tight project management, it should be possible to move up in this duration. It was also agreed that the head of the SEPG will be the project leader for this project with support and co-operation being provided, on a need basis, by the project people.

Besides satisfying a basic requirement for operating in a project mode, this setting of a target provided other benefits as well.

- It stopped all debates about the usefulness of CMM and the focus shifted to how to use it best for our business and environment.
- The task of deciding what SPI initiatives to undertake became much simpler as the framework was used.
- An immediate buy-in that followed when the senior management set level 4 as a corporate objective providing a shared goal among the various stake holder.
- Finding people to help in the SPI initiatives derived from the goal became a lot easier, as it was a corporate objective.
- As the initiative ended in assessment, it provided a relatively quick feedback on the project and, in our case, provided a great sense of achievement.



LEVERAGING EXISTING PRACTICES

When an organization follows or complies with some standard or framework, it deploys some practices and develops some structures to support the framework. It stands to reason that when a different framework is to be employed, the current practices and structures should be leveraged to the fullest. We established a guiding principle that for implementing the CMM framework, the existing structures and procedures that help implement ISO are to be re-used and leveraged as much as possible. Some of the structures and practices that an ISO organization is likely to have [8] are given below along with how they might be leveraged for CMM:

- Some quality system (QS) manuals or documents describing the various practices in the organization. The QS can be used for documenting all the processes. One particular document – the quality manual – which is often there in ISO organizations, can be used to document the various policies required by various KPAs in the CMM.
- A group within the organization, like a quality group or department. Such a group is likely to be doing process activities and hence can play the role of SEPG.
- An internal audit program, which requires that different aspects of implementation of the quality system of the organization are audited by some people who are independent from the ones doing the implementation. The audit mechanism can be effectively used to implement the Software Quality Assurance KPA of level 2, as well as for implementing key practices of different KPAs that require that some activity be independently reviewed or audited.

□ Some procedures to identify reasons for non-compliance, and for changing the processes, and disseminating the changes. The structures and mechanisms being used for this can be used for implementing processes and process changes to satisfy some requirements of the Organization Process Focus KPA.

□ A senior management review of activities of the quality group. This review can be expanded to implement those requirements of Organization Process Focus and Organization Process Definition KPAs which require senior management involvement.

□ Documentation policies and guidelines which require that all identified work products are documented. The implementation of this can be enhanced to satisfy requirements of various KPAs which require documentation of project plan, schedule, requirements, test plans, etc.

□ Documentation control procedures that ensure that proper versioning is maintained, the documents are reviewed and approved, and that impact of change of a document is understood and made on other documents as well. These mechanisms can be enhanced to handle some key practices of the Requirements Management KPA. The review practices for documents (which include plans as well as work products) can be enhanced to implement the Peer Review KPA.

□ An overall development life cycle specifying the major phases including requirements, design, coding, testing, and installation. This definition can be enhanced to a suitable degree of detail for the CMM. This definition can also be enhanced to support process tailoring, a requirement of the OPD KPA.

□ Project planning policies requiring that a proper project plan be developed which contains the estimates etc. before the development begins. The project plan, and the planning process, can be enhanced to implement the various requirements of the various KPAs.





- Configuration management policies and practices, which can be used to satisfy the Configuration Management KPA.
- Some training program with records of training being maintained. This can be enhanced to satisfy the Training Program KPA of level 3.
- Some metrics program. In particular, reporting of defects, and their tracking to closure. This program can be enhanced to collect more data that is needed for higher levels. The usage of metrics also will need to be refined considerably.
- The practices of maintaining quality records can be enhanced to collect defect data. Furthermore, analysis of such data can be used for quantitative process management.

These are some of the examples of how existing structures and practices that are in place in an ISO organization can be used for implementing the CMM. Besides trying to reuse existing structures, we also established the principle of simplicity – the processes should be kept simple and not very detailed or complicated. Our experience earlier with detailed processes was that they tend to put-off practitioners and are not amenable to “validation” (how do you ensure that the process is being followed?), and hence tended to remain “on-paper” processes. We also established that even though CMM is to be implemented, relevance of our processes to our business needs should not be sacrificed. This essentially meant that where it was perceived that some CMM requirement has limited value to our business, we needed to interpret it in a manner that it adds value to us, or do a very “light weight” implementation. As CMM allows a fair amount of flexibility in interpretation, by this approach the basic objectives of processes – to support the business effectively – is not transformed to an expedient objective of “achieve level”. This also provides a solid reason for undertaking the SPI initiative and makes it easier to get the required “buy-in” from the project people.

GAP ANALYSIS

The first step in executing the project was to identify the gaps in the existing processes with respect to CMM level 4. Some general purpose studies have been done to help this activity [8, 10]. A summary of possible gaps in an ISO organization with respect to level 3 and 4 of the CMM is given below [8].

| KPA | Probable Gaps |
|---------------------------------|--|
| Organization Process Focus | <ul style="list-style-type: none"> Method to identify and disseminate usage of new tools and processes that are already being used in some parts of the organization. Plan for software process development and improvement activities |
| Organization Process Definition | <ul style="list-style-type: none"> Documented procedure for developing and maintaining a process. Tailoring guidelines Process definition with sufficient details. Organization software process database Library of process assets |
| Training Program | <ul style="list-style-type: none"> Procedure for conducting training Course material preparation standards Waiver procedure |



| KPA | Probable Gaps |
|---------------------------------|---|
| Integrated Software Management | <ul style="list-style-type: none"> Tailoring guidelines Learning technical and management lessons Guidelines/procedure for risk management Tracking of effort, critical resources, etc. Thresholds for variation of actual performance on a project as compared to planned for taking action |
| Software Product Engineering | <ul style="list-style-type: none"> Rationale for tool selection Defect data analysis |
| Intergroup coordination | None |
| Peer Reviews | All activities and goals of this KPA |
| LEVEL 4 KPAs | |
| Quantitative Process Management | <ul style="list-style-type: none"> Methods for quantitatively managing a project, including making plans, collecting data and analyzing them, and taking corrective actions when necessary. Process capability in quantitative terms (and this capability used in project planning and execution) |
| Software Quality Management | <ul style="list-style-type: none"> Methods for setting quantitative quality goals for a project, methods for quantitatively monitoring the progress and taking corrective actions when necessary. Quality capability of the process known in quantitative terms |

The actual gaps in the processes of an organization will, obviously, depend on the exact nature of the processes. The general purpose gap-analysis can help in identifying the specific gaps. At Infosys, the gap analysis was done primarily by a group in SEPG. The main approach was to go over all the key practices in all the KPAs and identify what were the missing elements. This gap analysis was analogous to the requirements analysis phase in that it clearly defined the scope of the project and what was to be implemented. Initially, only gaps at a high level were identified. Some of the gaps we found were:

Risk Management – existing risk management was ad hoc and there was no established process for this.

Process tailoring – existing guidelines were minimal.

Peer reviews – existing reviews were un-structured, and mostly one-person reviews.

Estimation – Data from past experience was being used only informally.

Process database – There was no on-line process database, though project closure analysis was being done.

Quantitative project tracking – No guidelines in place.

Quantitative quality management – No guidelines in place.

Besides these, there were some small gaps relating to configuration management, usage of tools, knowledge sharing, etc



MANAGING THE PROJECT

The gap analysis indicated that there were about 8-10 areas that needed more attention, most of them related to project management, and many of them requiring formalization of existing approaches. As time was short, we decided to follow the “big bang” approach. That is, to define all the enhancements needed to reach level 4, validate them, and then deploy them all together. This is contrary to the commonly held belief that process changes should be done gradually. However, we felt that this big-bang approach was better suited. It not only reduces the cycle time, it also considerably reduced the training needs (otherwise every time changes are made, training will be needed), and kept the focus on the process initiative. However, it was clear that if this big bang approach is to succeed, the project of transitioning from ISO to level 4 will have to be planned and managed effectively. Here we discuss some aspects of planning and project management.

PROJECT PLANNING

Once the gap analysis was completed we knew what needed to be done to upgrade the processes to level 4 (i.e. the “requirements” were known). A project plan was made, which specified the tasks in the project, who it was assigned to, the start date, and the end date. The plan was maintained as a Microsoft Project Plan (MPP) document.

The plan divided the project into three logical phases – Phase I to define the processes, Phase II to deploy the processes, and Phase III for assessment. Initially, only details of the first phase were specified in the project plan; other phases were specified only at a high level (specifying the duration for each). Later, detailed activities for the other stages were also specified. Finally, the MSP schedule for this project had about a 100 schedulable tasks in it.

DEFINING THE PROCESSES

To define process for the gaps, the working group or task force approach was followed. For each major gap, a working group was



formed, which consisted of about 4 - 5 people, and one representative from the SEPG. In most groups, the group leader was some experienced person from projects.

Each working group was given specific instructions, where possible, about what is the nature of expected output. They were also sensitized to our guiding principles. The relevant KPAs that were the cause of the gaps were also pointed out. Their charter was to define (or enhance) the process to plug the gap and satisfy the relevant KPA, pilot the processes on one or two projects, prepare the necessary training material, and then hand it all over to SEPG for full deployment and training.

About 8 working groups worked in parallel. Due to our guiding principles, in most cases the scope of work was not large (due to our leveraging principle, little R&D had to be done). Each group was given about two to three months to finish their task. Most people in the groups, except the SEPG member, worked part-time, spending about one third of their time on this. Within about three months all the processes had been defined, and most piloted or tried on same existing projects.

PROCESS DEPLOYMENT

Deployment is always the hardest task of an SEPG. The two primary activities in a process-based approach are process definition and process deployment. Frequently, the focus of SEPG is on process definition, in the belief that that is the main task. Though process definition is a technically and intellectually a challenging task, a considerable effort is needed for process deployment. In this case, as this particular initiative was a corporate goal, the task of deployment had become easier. We decided to deploy these processes mostly in new projects, and let the existing projects continue to use old processes. This simplified the deployment task considerably as new projects come in with a steady, slow, rate, which permits the SEPG to “hand hold” them comfortably even with the limited staff.

First a massive training drive was embarked upon. Here we took the view that project people do not need to know CMM – they need to know our processes only. It is the task of the SEPG to ensure that our processes were “level 4 compliant”. Hence, the focus of our training was our processes. We packaged the training material in two programs – one for peer reviews as this was a new





process step that was being added with a new procedure, and the other was a comprehensive program having short sessions on each of the new process elements. While the latter training program targeted project leaders only as most of the new process concerned project management, peer reviews training was given to developers as well. Within a few weeks a large number of people has been imparted the training, as training programs were held multiple times every week.

To aid deployment, we also enhanced the role of the SEPG member who is usually associated with the project. The usual practice was that a SEPG member is associated with a project, who does not take part directly in any project activities but aids the projects is process definition and monitors the project for process compliance. As some of new process elements required the use of process database and data analysis periodically, we decided that support for this will be provided by the SEPG. Besides this, a checklist was prepared for the SEPG role in projects to make sure that processes are being properly implemented.

PROJECT MONITORING

Though proper monitoring is critical in any project, in this project monitoring and control were even more important. There were some reasons for this. First, the people in the working group were not members of the SEPG and hence did not report to the SEPG manager and had other tasks assigned to them too. Without direct control and influence on them by SEPG, proper control mechanisms became very important. Secondly, the plan for this project depended on the situations in other projects and hence plan became more dynamic, particularly in Phases II and III. Finally, as the project was to achieve a corporate goal, the stakes were much higher.

To ensure that all working groups deliver in time, even though SEPG was responsible for this project, we formed a steering team comprising of many senior manager and headed by the CEO. The steering team met once a month and all working groups reported their progress in this meeting. So, though regular monitoring was being done by SEPG, this provided the necessary “push” and commitment. Having the CEO as its



chair, right messages went out that the organization is serious about the initiative, which is an important success parameter [2, 3]. It also provided the visibility into the initiative to the senior management, another important success factor [2, 3]. Also, the steering team meetings and presentations gave the working groups a visibility to the CEO, which provided an added incentive to them, as the CEO ultimately controls the salary raises, promotions, stock options, etc.

RISK MANAGEMENT

Once a goal was set, and a tight deadline given, it was clear that there were many things that may go wrong during the project. In other words, there were many risks to this project. To handle this, effective risk management was needed. Risk management is typically not associated with SPI projects. However, we found that once the goal was clear, risk management was very useful in achieving our goal.

We followed a simple strategy for risk management. In each steering team meeting, particularly the early ones, we prepared a list of risks that we thought could adversely effect the outcome of the project. We ranked the risks as high, medium, or low. And for each high and medium risk we suggested some risk mitigation step.

The strategy of presenting and discussing risk management in the steering team was very useful. As many of the proposed risk mitigation steps were to be executed by people outside the SEPG, commitments were obtained during the steering team meeting. With the CMD and other senior managers in the steering team, agreeing and executing risk mitigation became considerably easier.

For example, as the early plan was to take only the development and re-engineering processes to level 4 , there was a risk that not enough such projects may volunteer (we had set a target that at least a dozen projects must be operating at level 4 by the assessment time.) As this was identified as a risk, during the steering team meeting itself projects got identified and committed, since most of the senior people were in the steering team. Another risk mitigation step for this was to expand the scope and include maintenance projects also. Again, during the steering





team meeting commitments were obtained and relevant working groups formed for maintenance.

Another risk was number of entries in the process database. We had set a target that there must be about 50 entries in the process database within the next six months. From the old project closure reports, we were able to add about 20 data points. Hence, the rest had to come from recently completed projects or projects that were to complete in the near future. This was proposed as the risk mitigation strategy. Once this was accepted, all such projects were asked to furnish their project completion analysis reports, which was a somewhat hard task earlier.

One risk we identified early was “difference in interpretation with the assessor”. We had interpreted and implemented the CMM in some manner, which we thought suited us the best. However, how were we to be sure that this interpretation will be accepted by the assessor. This is a very real risk, particularly since CMM does not prescribe processes and so different assessors may take a different view of same aspects of CMM. Non-prescriptive nature of CMM gives it the flexibility but also makes the task of implementing CMM harder [2, 3]. For risk mitigation, we employed a CMM consultant for a few days to evaluate our processes, not their implementation, with respect to the CMM.

CO-EXISTANCE OF ISO AND CMM

After an organization has been assessed at a high maturity level of the CMM, it has to decide what it wants to do with its ISO certification. There are two clear options available – relinquish the ISO certification, or continue with both ISO and the CMM.

If the ISO structures are leveraged effectively for implementing the CMM framework, then there is no conflict between the two frameworks, and satisfying two frameworks does not “double the paper work”. In other words, there is no need to relinquish ISO. At Infosys, as we used the ISO structures fully, maintaining ISO actually became easier after attaining the level 4 of the CMM. In fact, the external audits that ISO has, helped in strengthening the deployment of the CMM.



Overall, we found that ISO and CMM are consistent with each other and can peacefully co-exist in an organization.

SUMMARY

There is now enough evidence to suggest that software process improvement can provide good returns to any organization developing software. In an effort to further improve its processes, Infosys decided to move from ISO to higher levels of the CMM. Within one year of starting the transitioning initiative, it was assessed at level 4 of the CMM. In this article, we discussed various elements of the strategy we followed for this rapid climb up the maturity ladder, which we believe are quite general and reusable in other contexts also. Some of the important elements of the strategy were:

- Setting a goal for the transitioning in terms of a maturity level. This provided a clear and succinct objective which will necessarily require commitment from senior management. Setting an “icon” makes it easier to rally the organization around the “icon”. It also provides an easy way of validating whether goals of the project have been achieved – through an assessment.
- Keep the duration of the project not too long. This allowed keeping the focus and interest and a relatively quick feedback on the effort. In a short duration, the processes will get defined and the capability for the particular maturity level will get developed. Spreading the processes can continue to take place after the successful completion of the project as part of regular SEPG activities.
- Principles of leveraging the existing structures and simplicity help in reducing the “amount of change” and getting processes that are likely to be followed.
- The transitioning initiative with the goal as a maturity level should be managed like a project. Proper planning should be done, followed by proper implementation and monitoring. It helps to have monitoring done by senior management as it provides them visibility into the project and preserves their commitment and support for the SPI initiative.





- Risk management techniques can be very effective in ensuring that the goals of the SPI project are met. We used risk management very effectively to continually reduce the risk of failure.
- ISO and CMM can co-exist peacefully without extra overhead.

REFERENCES

1. M. Daskalantonakis, "Achieving higher SEI levels", *IEEE Software*, July 1994, pp. 17-24.
2. J. D. Herbsleb and D.R. Goldenson, "A systematic survey of CMM experience and results", *18th Int. Conf. On Software Engineering*, Berlin, 1996, pp. 323-330.
3. J. Herbsleb et. al., "Software quality and the capability maturity model", *Commn. Of the ACM*, 40:6, June 1997, pp. 31-40.
4. W. Humphrey, *Managing the Software Process*, Addison Wesley, 1989.
5. *ISO9001, Quality Systems – Model for Quality Assurance in Design/Development, Production, Installation, and Services*, Intl. Standards Organization, Geneva, 1987.
6. *ISO9000-3: Guidelines for the application of ISO9001 to the development, supply and maintenance of software*, International Standard, 1991.
7. *TickIT: A Guide to Software Quality Management System Construction and Certification Using EN29001*, UK Dept. of Trade and Industry and British Computer Society, 1992.
8. P. Jalote, Moving from ISO to higher levels of the CMM, *SEPG Conference*, Atlanta, 1998.
9. M. Paulk, et. al., "*The Capability Maturity Model for Software, Version 1.1*", CMU/SEI-93-TR-24, Software Engineering Institute, 1993, also published as book by Addison Wesley, 1995.
10. M. C. Paulk, "Comparing ISO 9001 and the capability maturity model for software", *Software Quality Journal*, vol 2, 1993, pp. 245-256.
11. K. Sakamoto, et. al., "Toward computational support for software process improvement activities", *Proc. 20th Int. Conf. On Software Engg.*, 1998, pp. 22-31.
12. [Maturity profile report from SEI, www.sei.cmu.edu/sema/profile.html.3.download.html](http://www.sei.cmu.edu/sema/profile.html.3.download.html)



ISO to CMM

Managing the Transition from ISO to High Maturity Levels of the CMM

K. Dinesh
Infosys Technologies Ltd.
Bangalore

Co-Authors:
P. Jalote, M. R. Bhashyam, S. Raghavan

About Infosys

- Software house with over 3000 engineers
- 15 locations world-wide
- Customers in 6 countries
- 60% annual growth for the last 5 years
- Multiple offerings - development, maintenance, reengineering, Y2K,
- Different business domains



Infosys - Quality Journey

- Before 1991 - some processes; performance varied a lot depending on people
- 1993 - got ISO9001 (TickIT) certification
 - One of the first software companies to do so
 - Helped in ensuring uniformity
 - Better control on projects
 - Extremely useful in managing the huge growth
- ISO an excellent starting point in quality journey; established process-oriented culture

Culture Change with IS O

Before ISO

- Fire fighting and crisis management
- Wide variation in QP
- Little control and visibility
- Focus on short term gains and quick fixes
- Personality orientation

Post ISO

- Planned project execution
- Better control and visibility
- Customer orientation
- Focus on long term
- Involved entire organization in standardization

Why CMM?

- ISO has only 2 levels - cannot differentiate an organization that has gone beyond
- Limited guidance for a graded approach to process improvement
- CMM was becoming very popular, and many US customers were demanding it
- Many competitors were adopting CMM
- Need for quantitative visibility in quality and productivity and quant. control

What is CMM?

- CMM categorizes process maturity in 5 levels - level 1 (lowest) to level 5 (highest)
- Only about 40 organizations world-wide at level 4 and 5

CMM and ISO

| Level | Focus | Key Process Areas | ISO9000 |
|----------------|--------------------------------|--|--|
| 5 - Optimizing | Continuous Process Improvement | Defect prevention Technology change mgmt. Process change mgmt. | X |
| 4 - Managed | Product and Process Quality | Quantitative process mgmt. Software quality mgmt. | X |
| 3 - Defined | Engineering process | Organization process focus Organization process defn. Training program Integrated software mgmt. Software product engineering Intergroup coordination Peer reviews | X(SEPG) X(Process Assets) X(Well defined) |
| 2 - Repeatable | Project Management | Requirements management Software project planning Software project tracking Software subcontract mgmt. Software quality assurance Software configuration mgmt. | Replication, Delivery and Installation Customer maintenance agreement, Purchasing |
| 1 - Initial | | | |

S strategy for T ransition

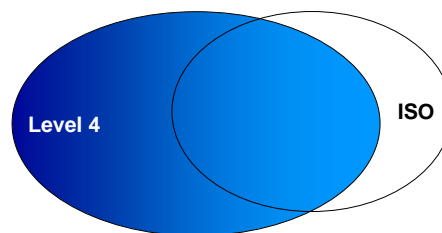
- Leverage existing ISO structures and practices so that there is no redundancy
- Manage transition like an aggressive project
- Set a corporate goal in terms of level (4)
- Have a high-powered steering team to monitor the project and keep tight control
- Keep a tight schedule to preserve focus

Why Level 4?

- An ISO organization satisfies most level 2 requirements and some level 3 req.
- Lowest level to shoot for is level 3
- Level 4 provides the quantitative visibility and control; has 2 more KPAs
- Suitable target level is 4 - is ambitious and brings about next level of cultural change (metrics orientation)

Leverage Existing Practices

For improvement, structures for different frameworks should be seamless



Most ISO structures were utilized; some new were added

SPI Project Planning

- Decided to follow the big-bang approach for process enhancement
- Three phase plan
 - Process definition/refinement
 - Process deployment
 - Assessment

Process Definition/Refinement

- Did gap analysis of existing processes with respect to level 4 requirements
- About 8 major gaps; many minor gaps
 - Process tailoring, risk management, peer reviews, estimation, process database, quantitative process and quality management
- Formed work groups to enhance processes; they worked in parallel
 - Process implementation infrastructure existed. Preparation for close to 2 years
 - In the final stretch of 3 to 5 months, processes were refined/enhanced and most piloted

Process Deployment

- **Massive training drive**
 - Project mgmt changes - to project leaders
 - Inspections - to developers also
- **SEPG help to projects enhanced**
- **Only new projects to implement new processes; old ones continued**
 - Helped make implementation smooth and gradual

Assessment

- **Internal assessment through key-practice wise check list**
- **Internal audit focus areas made consistent with the CMM initiative**
- **Abridged process assessment done**
- **Then final assessment done**

SPI Project Monitoring

- Work groups involved over 50 people from different groups, most working part-time
- Not in direct control of SEPG
- Formed a steering team of very senior people for monitoring the project
 - Met monthly to review progress
 - Resolved issues that needed senior mgm. Input
 - Made sr. mgmt. Partner in process improvement

Monitoring - Risk Management

- An aggressive project has risks - employed risk management effectively for this
- Identified risks, prioritized them, and chose their mitigation steps
- Presented in steering team meetings and got commitments for mitigation
- Some risks: not enough data points; not enough development projects; interpretation

Roles and Responsibilities



Key Challenges

- Developing “metrics orientation”
 - Capture of project performance data - process database
 - Conceiving and building process capability baseline
 - Using statistical methods for project control and handling performance variation

Key Challenges - Contd.

- **Institutionalizing Inspections**
 - Group reviews are counter intuitive: how can review be better than testing?
 - Are perceived to be very expensive
 - Data from others not believed
 - Not Applicable Here (NAH) syndrome needs to be handled
 - Did experimentation and massive training

Key Challenges - Contd.

- **Change management - the people aspect**
 - Need leaders, champions, early adopters
 - SEPG took the leadership
 - Champions were pilots, working gp. members
 - Early adopters as projects
 - Middle management is the key

Lessons Learned

- **Aggressive schedule possible only if strong process implementation infrastructure exists**
- **Frameworks have enough flexibility to match business needs**
- **Senior management support and commitment is essential**
- **Setting a goal in terms of a level helps rally the organization**

Lessons Learned - Contd.

- **Backward planning from end date was effective**
- **Keeping processes simple, light-weight, and practical helped acceptance**
- **Keeping the metrics set simple and realistic helped in deployment**
- **Middle management is the critical link**
- **Good information infrastructure is essential**

Lessons Learned - Limitations

- Should have viewed level 5 as the ultimate goal and level 4 as the immediate goal
- Metrics usage and analysis could be done more effectively

ISO to Level 4 - Cultural Change

ISO

Troubleshooting
Fire-fighting
Crisis Management
Focus on:
Short-Term Gains
Customer Complaints
Quick Fixes

Level4(+)

Quantitative process
management
Quantitative control
Better planning
Goal directed improvement
Higher predictability
Better understanding and
control of risks
Involvement of senior
management in process
definition

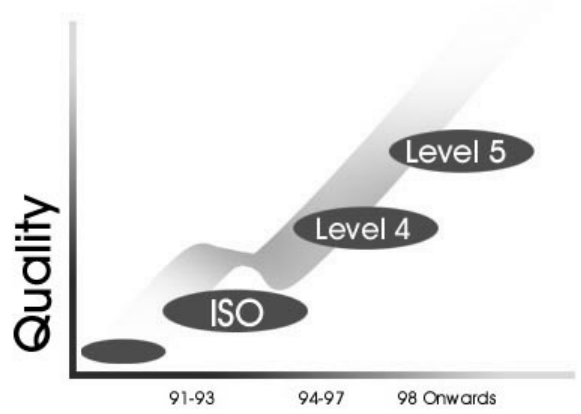
ISO and CMM Co-Existence

- Is having CMM and ISO not double work?
- **No. Both can co-exist; Infosys keeps both**
 - Structures are used for both
 - Processes are common
 - So, no redundancy; CMM becomes a “super set” of the ISO practices/structures

Summary

- Transition from ISO to CMM is smooth, level 4 seems to be the right target
- **Managing it as an aggressive project has many benefits**
- **Senior management monitoring helps**
- **Plenty of benefits**
- **Is not the final destination in the quality journey!**

The Journey Continues



Thanks
Thank you

IMPROVEMENT OF THE TEST PROCESS

using TPI[®]

T.Koomen, M. Pol
IQIP Software Control Testen

P.O. Box 263, 1110 AG Diemen
The Netherlands

Tel: +31 20 6606600

Fax: +31 20 6953298

e-mail: tpi@iquip.nl

English TPI[®] website: www.iquip.nl/tpi

Abstract

This paper presents the TPI[®]-model, which is based on current state-of-the-art test process improvement practices. The model gives practical guidelines for assessing the maturity level of testing in an organisation and for step by step improvement of the process. The purpose of such improvement could be reaching CMM level 3.

The model consists of 20 key areas, each with different levels of maturity. The levels of all key areas are set out in a maturity matrix. Each level is described by several checkpoints. Improvement suggestions, which help to reach a desired level, are part of the model.

The paper includes a general description of the application of model, which deals with how to implement and how to consolidate the improvements.

1 How good is your test process ?

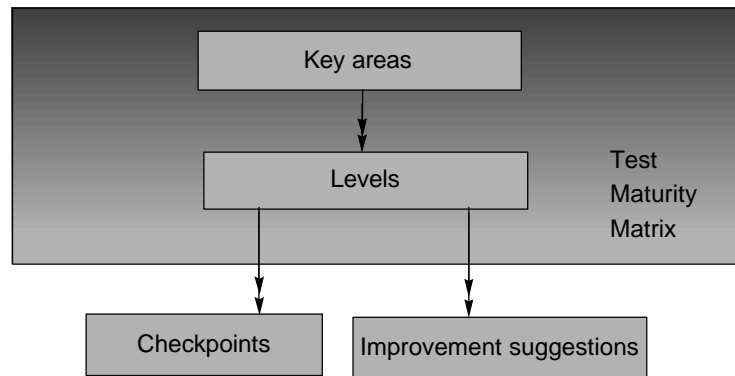
This seemingly easy question turns out to be very hard to answer in reality. Testing is often experienced as a troublesome and uncontrollable process. Testing takes too much time, costs a lot more than planned, and offers insufficient insight in the quality of the test process and, therefore, the quality of the information system under test and the risks for the business process itself. But can we do something about this ?

Many organisations realise that improving the test process can solve these problems. However, in practice it turns out to be hard to define what steps to take for improving and controlling the process, and in what order. A comparison can be made with improvement of the total software process, where models like the Capability Maturity Model[®] (CMM) offer support.

Based on the knowledge and experiences of a large number of professional testers the Test Process Improvement (TPI[®]) model has been developed. The TPI model supports the improvement of test processes. The model offers insight in the "maturity" of the test processes within your organisation. Based on this understanding the model helps to define gradual and controllable improvement steps.

2 Description of the model

The model is visualised as follows:



Key areas

In each test process certain areas need specific attention in order to achieve a well defined process. These **Key areas** are therefore the basis for improving and structuring the test process. The TPI model has 20 key areas.

The scope of test process improvement usually comprises high-level tests like system and acceptance tests. Most key areas are adjusted to this. However, to improve more "mature" test processes, attention must also be given to verification activities and low-level tests like unit and integration tests. Separate key areas are included in order to give due attention to these processes as well.

A full list of key areas is given below, followed by an explanation.

| | | |
|-------------------------------|--------------------------------|-------------------------|
| Test strategy | Test environment | Defect management |
| Life-cycle model | Office environment | Testware management |
| Moment of involvement | Commitment and motivation | Test process management |
| Estimating and planning | Testing functions and training | Evaluation |
| Test specification techniques | Scope of methodology | Low-level testing |
| Static test techniques | Communication | |
| Metrics | Reporting | |
| Test tools | | |

| Key area | Description |
|-------------------------------|--|
| Test strategy | The test strategy has to be focused on detecting the most important defects as early and as cheaply as possible. The test strategy defines which requirements and (quality) risks are covered by what tests. The better each test level defines its own strategy and the more the different test level strategies are adjusted to each other, the higher the quality of the overall test strategy. |
| Life-cycle model | Within the test process a number of phases can be defined, such as planning, preparation, specification, execution and completion. In each phase several activities are performed. For each activity the following aspects should be defined: purpose, input, process, output, dependencies, applicable techniques and tools, required facilities, documentation, etc.. The importance of using a life-cycle model is an improved predictability and controllability of the test process, because the different activities can be planned and monitored in mutual cohesion . |
| Moment of involvement | Although the actual execution of the test normally begins after the realisation of the software, the test process must and can start much earlier. An earlier involvement of testing in the system development path helps to find defects as soon and easy as possible and perhaps even to prevent errors. A better adjustment between the different tests can be done and the time that testing is on the critical path of the project can be kept as short as possible. |
| Estimating and planning | Test planning and estimating indicate which activities have to be carried out when, and the necessary resources (people). Good estimating and planning are very important, because they are the basis of, for example, allocating resources for a certain time frame. |
| Test specification techniques | The definition of a test specification technique is "a standardised way of deriving test cases from source information". Applying these techniques gives insight into the quality and depth of the tests and increases the reusability of the test. |
| Static test techniques | Not everything can and should be tested dynamically, that is, by running programs. Inspection of products without running programs, or the evaluation of measures which must lead to a certain quality level, is called static tests. Checklists are very useful for this. |
| Metrics | Metrics are quantified observations of the characteristics of a product or process. For the test process, metrics of the progress of the process and the quality of the tested system are very important. They are used to control the test process, to substantiate the test advice and also to make it possible to compare systems or processes. Why has one system far fewer failures in operation than another system, or why is one test process faster and more thorough than another? Specifically for improving the test process, metrics are important by evaluating consequences of certain improvement actions, by comparing data before and after performing the action. |
| Test tools | Test tools are automated aids for the test process. Automation within the test process can take place in many ways and has in general one or more of the following aims: <ul style="list-style-type: none"> - fewer hours needed, - shorter lead time, - more test depth, - increased test flexibility, - more and/or faster insight in test process status, - better motivation of the testers. |
| Testing environment | The test execution takes place in a so-called test environment. This environment mainly comprises the following components: <ul style="list-style-type: none"> - hardware; - software; - means of communication; - facilities for building and using databases and files; - procedures. |

| | |
|--------------------------------|--|
| | The environment should be composed and set up in such a way that by means of the test results it can be optimally determined to what extent the test object meets the requirements. The environment has a large influence on the quality, lead time, and cost of the test process. Important aspects of the environment are responsibilities, management, on-time and sufficient availability, representativeness, and flexibility. |
| Office environment | The test staff need rooms, desks, chairs, PCs, word-processing facilities, printers, telephones, and so on. A good and timely organisation of the office environment has a positive influence on the motivation of the test staff, on communication in- and outside the team, and on the efficiency of the work. |
| Commitment and motivation | The commitment and the motivation of the persons involved in testing are important prerequisites for a smoothly running test process. The persons involved are not only the testers, but also, for example, the project management and the line management personnel. The latter are mainly important in the sense of creating good conditions. The test process thus receives enough time, money, and resources (quantitatively and qualitatively) to perform a good test, in which cooperation and good communication with the rest of the project results in a total process with optimum efficiency. |
| Testing functions and training | In a test process the correct composition of a test team is very important. A mix of different disciplines, functions, knowledge, and skills is required. Besides specific test expertise, knowledge of the subject matter, knowledge of the organisation and general IT knowledge is required. Social skills are also important. For acquiring this mix, training etc. is required. |
| Scope of methodology | For each test process in the organisation a certain methodology or working method is used, comprising activities, procedures, regulations, techniques etc.. When these methodologies are different each time or when the methodology is so generic that many parts have to be drawn up again each time, it has a negative effect on the test process efficiency. The aim is that the organisation uses a methodology which is sufficiently generic to be applicable in every situation, but which contains enough detail so that it is not necessary to rethink the same items again each time. |
| Communication | In a test process, communication with the people involved must take place in several ways, within the test team as well as with parties such as the developer, the user, the customer, etc.. These communication forms are important for a smoothly running test process, not only to create good conditions and to optimize the test strategy, but also to communicate about the progress and the quality. |
| Reporting | Testing is not so much "defect detection" as about giving insight in the quality level of the product. Reporting should be aimed at giving well-founded advice to the customer concerning the product and even the system development process. |
| Defect management | Although managing defects is in fact a project matter and not specifically of the testers, the testers are mainly involved in it. Good management should be able to track the life-cycle of a defect and also to support the analysis of quality trends in the detected defects. Such analysis is used, for example, to give well-founded quality advice. |
| Testware management | The products of testing should be maintainable and reusable and so they must be managed. Besides the products of the testing themselves, such as test plans, specifications, databases and files, it is important that the products of previous processes such as functional design and realisation are managed well, because the test process can be disrupted if the wrong program versions, etc. are delivered. If testers make demands upon version management of these products, a positive influence is exerted and the testability of the product is increased. |
| Test process management | For managing each process and activity, the four steps from the Deming circle are essential: plan, do, check and act. Process management is of vital importance for the realisation of an optimal test in an often turbulent test |

| | |
|-------------------|--|
| | process. |
| Evaluation | Evaluation means inspecting intermediate products such as the requirements and the functional design. The importance of evaluation is that the defects are found at a much earlier stage in the development process than with testing. This makes the rework costs much lower. Also, evaluation can be set up more easily because there is no need to run programs or to set up an environment etc.. |
| Low-level testing | The low-level tests are almost exclusively carried out by the developers. Well-known low-level tests are the unit test and the integration test. Just as evaluation, the tests find defects at an earlier stage of the system development path than the high-level tests. Low-level testing is efficient, because it requires little communication and because often the finder is both the error producer as well as the one who corrects the defect. |

Levels

The way key areas are organised within a test process determines the 'maturity' of the process. It is obvious that not each key area will be addressed equally thoroughly: each test process has its strengths and weaknesses.

In order to enable insight in the state of the key areas, the model supplies them with **Levels** (from A to B to C). On the average, there are three levels for each key area.

Each higher level (C being higher than B, B being higher than A) is better than its prior level in terms of time (faster), money (cheaper) and/or quality (better). By using levels we can unambiguously assess the current situation of the test process. It also increases the ability to advice targets for stepwise improvement.

Each level consists of certain requirements for the key area. The requirements (= checkpoints) of a certain level also comprise the requirements of lower levels: a test process at level B fulfils the requirements of both level A and B. If a test process does not satisfy the requirements for level A, it is considered to be at the lowest and, consequently, undefined level for that particular key area.

Below a description is given of the different levels of the key areas.

| Levels | A | B | C | D |
|-----------------------------|--|---|---|--|
| Key area | | | | |
| Test strategy | Strategy for single high-level test | Combined strategy for high-level tests | Combined strategy for high-level tests plus low-level tests or evaluation | Combined strategy for all evaluation levels |
| Life-cycle model | Planning, Specification, Execution | Planning, Preparation, Specification, Execution, Completion | | |
| Moment of involvement | Completion of test basis | Start of test basis | Start of requirements definition | Project initiation |
| Estimating and planning | Substantiated estimating and planning | Statistically substantiated estimating and planning | | |
| Design techniques | Informal techniques | Formal techniques | | |
| Static test techniques | Inspection of test basis | Checklists | | |
| Metrics | Project metrics (product) | Project metrics (process) | System metrics | Organisation metrics (>1 system) |
| Test tools | Planning and control tools | Execution and analysis tools | Extensive automation of the test process | |
| Test environment | Managed and controlled environment | Testing in most suitable environment | Environment on call | |
| Office environment | Adequate and timely office environment | | | |
| Commitment and motivation | Assignment of budget and time | Testing integrated in project organisation | Test-engineering | |
| Test functions and training | Test manager and testers | (Formal) Methodical, technical and functional support, management | Formal internal Quality Assurance | |
| Scope of methodology | Project specific | Organisation generic | Organisation optimising (R&D) | |
| Communication | Internal communication | Project communication (defects, change control) | Communication within the organisation about the quality of the test processes | |
| Reporting | Defects | Progress (status of tests and products), activities (costs and time, milestones), defects with priorities | Risks and recommendations, substantiated with metrics | Recommendations have a Process Improvement character |
| Defect management | Internal defect management | Extended defect management with flexible reporting facilities | Project defect management | |
| Testware management | Internal testware management | External management of test basis and test object | Reusable testware | Traceability system requirements |
| Test process management | Planning and execution | Planning, execution, monitoring, and adjusting | Monitoring and adjusting within organisation | |
| Evaluation | Evaluation techniques | Evaluation strategy | | |

| | | | | | |
|-------------------|--|-----------|----------------------|-------------------------|--|
| Low-level testing | Low-level test life-cycle: specification and execution | planning, | White-box techniques | Low-level test strategy | |
|-------------------|--|-----------|----------------------|-------------------------|--|

Checkpoints

In order to determine levels, the TPI model is supported by an objective measurement instrument. The requirements for each level are defined in the form of **Checkpoints**: questions that need to be answered positively in order to classify for that level. Based on the checkpoints a test process can be assessed, and for each key area the proper level can be established. As each next level of a key area is considered an improvement, this means that the checkpoints are cumulative: in order to classify for level B the test process needs to answer positively to the checkpoints both of level B and of level A.

Test Maturity Matrix

After determining the levels for each key area, attention should be directed as to which improvement steps to take. This is because not all key areas and levels are equally important. For example, a good test strategy (level A of key area Test Strategy) is more important than a description of the test methodology used (level A of key area Scope of Methodology). In addition to these priorities there are dependencies between the levels of different key areas. Before statistics can be gathered for defects found (level A of key area Metrics), the test process has to classify for level B of key area Defect management. Such dependencies can be found between many levels and key areas.

Therefore, all levels and key areas are related to each other in a **Test Maturity Matrix**. This has been done as a good way to express the internal priorities and dependencies between levels and key areas. The vertical axis of the matrix indicates key areas, the horizontal axis shows scales of maturity. In the matrix each level is related to a certain scale of test maturity. This results in 13 scales of test maturity. The open cells between different levels have no meaning in themselves, but indicate that achieving a higher maturity for a key area is related to the maturity of other key areas. There is no gradation between levels: as long as a test process is not entirely classified at level B, it remains at level A.

| | Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------------------------------|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Key area | | | | | | | | | | | | | | | |
| Test strategy | | | A | | | | | B | | | | C | | D | |
| Life-cycle model | | | A | | | B | | | | | | | | | |
| Moment of involvement | | | | A | | | | B | | | | C | | D | |
| Estimating and planning | | | | | A | | | | | | | B | | | |
| Test specification techniques | | | A | | B | | | | | | | | | | |
| Static test techniques | | | | | | A | | B | | | | | | | |
| Metrics | | | | | | | A | | | B | | | C | | D |
| Test tools | | | | | | A | | | B | | | C | | | |
| Test environment | | | | | A | | | | B | | | | | | C |
| Office environment | | | | | A | | | | | | | | | | |
| Commitment and motivation | | | A | | | | B | | | | | | C | | |
| Test functions and training | | | | | A | | | B | | | C | | | | |
| Scope of methodology | | | | | | A | | | | | | B | | | C |
| Communication | | | | A | | B | | | | | | | C | | |
| Reporting | | | A | | | B | | C | | | | | D | | |

| | Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------------------------|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Key area | | | | | | | | | | | | | | | |
| Defect management | | | A | | | | B | | C | | | | | | |
| Testware management | | | | A | | | B | | | | C | | | | D |
| Test process management | | | A | | B | | | | | | | | C | | |
| Evaluation | | | | | | | | A | | | B | | | | |
| Low-level testing | | | | | | A | | B | | C | | | | | |

The main purpose of the matrix is to show the strong and weak sides of the current test process and to support prioritising actions for improvement. A filled in matrix offers all participants a clear view of the current situation of the test process. Furthermore, the matrix helps in defining and selecting proposals for improvement.

The matrix works from left to right, so low mature key areas are improved first. As a consequence of the dependencies between levels and key areas, practice has taught us that real 'outliers' (i.e., key areas with high scales of maturity, whereas surrounding key areas have medium or low scales) give little return on investment. For example, what is the use of a very advanced defect administration, if it is not used for analysis and reporting? Without violating the model, deviation is permitted, but sound reasons should exist for it.

In the example below, the test process does not classify for the lowest level of the key area test strategy (level < A), the organisation is working conform a life-cycle model (level A) and the testers are involved at the moment when the specifications are completed (level A).

| | Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-----------------------|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Key area | | | | | | | | | | | | | | | |
| Test strategy | | | A | | | | | B | | | | C | | D | |
| Life-cycle model | | | A | | | B | | | | | | | | | |
| Moment of involvement | | | | A | | | | B | | | | C | | D | |
| etc. | | | | | | | | | | | | | | | |

Based on this instance of the matrix, improvements can be discussed. In this example, a choice is made for a combined test strategy for high-level tests (\Rightarrow level B) and for a full life-cycle model (\Rightarrow level B). Earlier involvement is at this moment not considered to be of relevance. The required situation is represented in the following matrix.

| | Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-----------------------|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| Key area | | | | | | | | | | | | | | | |
| Test strategy | | | A | | | | | B | | | | C | | D | |
| Life-cycle model | | | A | | | B | | | | | | | | | |
| Moment of involvement | | | | A | | | | B | | | | C | | D | |
| etc. | | | | | | | | | | | | | | | |

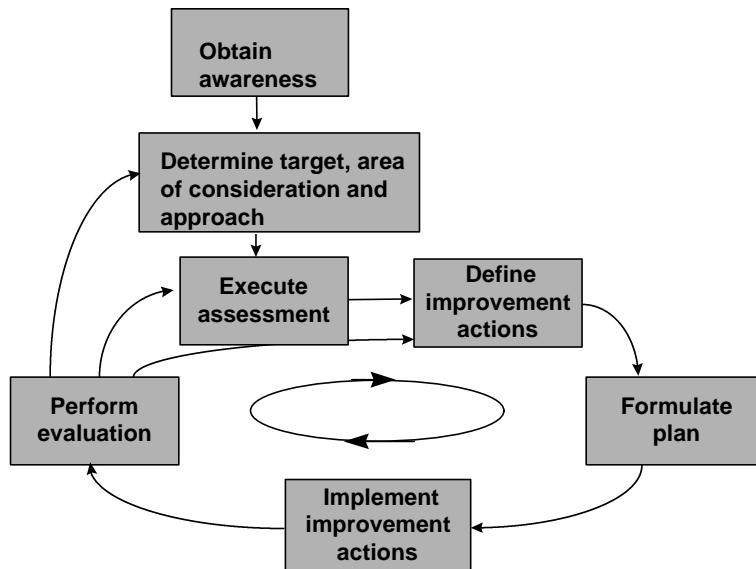
Improvement Suggestions

Improvement actions can be defined in terms of desired higher levels. For reaching a higher level the checkpoints render much assistance. Beside these, the model has other means of support for test process improvement: the **Improvement Suggestions**, which are different kinds of hints and ideas that help to achieve a certain level of test maturity. Unlike the use of

checkpoints, the use of improvement suggestions is not obligatory. Each level is supplied with several improvement suggestions.

3 Application of the TPI model

The process of test improvement is similar to any other improvement process. The figure below shows the various activities of an improvement process. These activities are discussed, with special attention for the places where the TPI model can be used.



Obtain awareness

The first activity of a test improvement process is to create awareness for the necessity to improve the process. Generally speaking, a number of problems concerning testing is the reason for improving the test process. There is a need to solve these problems and an improvement of the test process is regarded as the solution. This awareness also implies that the parties mutually agree on the outlines and give their commitment to the change process. Commitment should not only be acquired at the beginning of the change process, but be retained throughout the project. This requires a continuous effort.

Determine target, area of consideration, and approach

We determine what the improvement targets are and what the area of consideration is. Should testing be faster, cheaper or better? Which test processes are subjects for improvement, how much time is available for the improvement and how much effort is it allowed to cost?

Execute assessment

In the assessment activity, an evaluation is given of the current situation. The use of the TPI model is an important part of the assessment, because it offers a frame of reference to list the strong and weak points of the test process. Based on interviews and documentation, the levels per key area of the TPI model are examined by using checkpoints, and it is determined which checkpoints were met, which were not met, or only partially. The Test Maturity Matrix is used here to give the complete status overview of the test process. This will show the strengths and weaknesses of the test process in the form of levels assigned key areas and their relative position in the matrix.

Define improvement actions

The improvement actions are determined based on the improvement targets and the result of the assessment. These actions are determined in such a way that gradual and step by step improvement is possible.

The TPI model helps to set up these improvement actions. The levels of the key areas and the Test Maturity Matrix give several possibilities to define gradual improvement steps. Depending on the targets, the scope, the available time and the assessment results, it can be decided to carry out improvements for one or more key areas. For each selected key area it can be decided to go to the next level or, in special cases, even to a higher level. Besides this, the TPI model offers a large number of improvement suggestions which help to achieve higher levels.

Formulate plan

A detailed plan is drawn up to implement (a part of) the short term improvement actions. In this plan the aims are recorded and it is indicated which improvements have to be implemented at what time to realise these aims. The plan deals with activities concerning the content of the test process improvement as well as general activities needed to steer the change process in the right direction.

Implement improvement actions

The plan is executed. Because during this activity the consequences of the change process have the largest impact, much attention should be spent on communication. Opposition, which no doubt is present, must be brought to the surface and be discussed openly.

It has to be measured to what extent actions have been executed and have been successful. A means for this is the so-called "self assessment", in which the TPI model is applied in order to quickly determine the progress. Here, the persons involved inspect their own test processes using the TPI model.

Another vital part of this phase is consolidation. It should be prevented that the implemented improvement actions have a once-only character.

Perform evaluation

To what extent did the implemented actions yield the intended result? In this phase the aim is to see to what extent the actions were implemented successfully as well as to evaluate to what extent the initial targets were met. A decision about the continuation of the change process is made based on these observations.

4 Conclusions and remarks

Current developments proceed at a very high speed. The productivity of developers is rising continuously and the customers demand ever higher quality. Even if your current test process is fairly satisfactory, your process will need to improve in the future. The TPI model can help you with this.

The TPI model is an objective means to gain quick insight in the current situation of the test process. The model greatly offers help for improvement in the form of key areas, levels and improvement suggestions. It supports the definition of small and controlled improvement steps, based on priorities.

The reader might get the impression that use of the TPI model automatically leads to good analysis of the current and required situation. This is not true. The model should be seen as a tool for structuring the improvement of the test process and as a very good means of communication. Apart from the tool, improvement of test processes demands a high degree of knowledge and expertise of people involved, at least in the areas of testing, organisation and change management.

Book:

Koomen, T., Pol, M. (1999), Test Process Improvement, a practical step-by-step guide to structured testing, Addison-Wesley, ISBN 0 201 59624 5

Internet:

at 'www.iquip.nl/tpi' several TPI products can be viewed and downloaded. Also questions can be asked and remarks can be made.

Test Process Improvement Experiences: everything you always wanted to know ...

Tim Koomen
IQUIP Informatica BV
koomenti@iquip.nl



Agenda

- The TPI[®] model
- 3 TPI Experiences



Agenda

- • The TPI® model
- 3 TPI Experiences



What is Test Process Improvement?

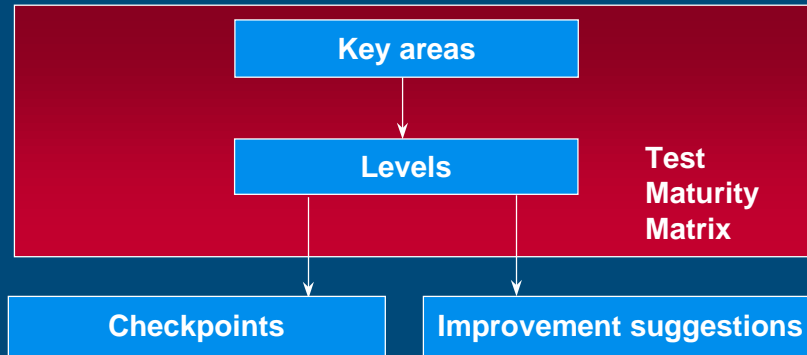
“Optimizing the quality, costs and lead time of the test process, in relation to the total information services”

- Quality
 - Insight
 - Coverage
 - Control
 - Timeliness
- Costs
 - Cheaper
- Lead time
 - Faster

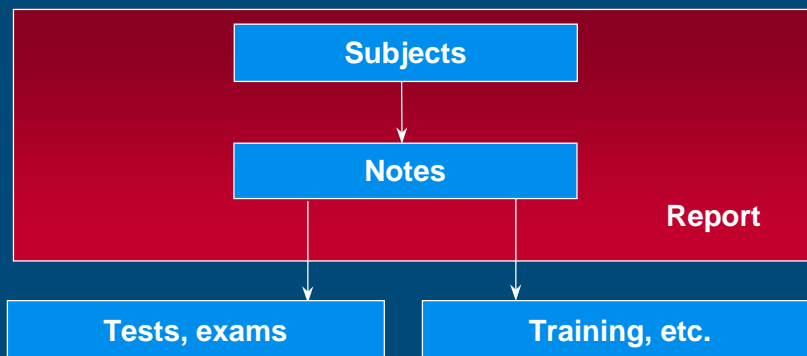
**Required:
a reference model**



The TPI[®] model



Metaphor TPI[®] model





Test Maturity Matrix

| Key Area / Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 1 Test strategy | | A | | | | | B | | | | C | | D | |
| 2 Life-cycle model | | A | | | B | | | | | | | | | |
| 3 Moment of involvement | | | A | | | | B | | | | C | | D | |
| 4 Estimating and planning | | | | A | | | | | | B | | | | |
| 5 Test specification techniques | | A | | B | | | | | | | | | | |
| 6 Static test techniques | | | | | | | | | | | | C | D | |
| 7 Metrics | | | | | | | | | | | | | | |
| 8 Test tools | | | | | A | | | B | | | | | | C |
| 9 Test environment | | | | A | | | | B | | | | | | C |
| 10 Office environment | | | | A | | | | | | | | | | |
| 11 Commitment and motivation | | A | | | | | | | | | | C | | |
| 12 Test functions and training | | | | A | | | B | | | | C | | | |
| 13 Scope of methodology | | | | | A | | | | | | B | | | C |
| 14 Communication | | | | A | | B | | | | | | C | | |
| 15 Reporting | | | A | | B | | C | | | | | D | | |
| 16 Defect management | | A | | | | B | | C | | | | | | |
| 17 Testware management | | | A | | | B | | | | C | | | | D |
| 18 Test process management | | A | | B | | | | | | | | C | | |
| 19 Evaluation | | | | | | A | | | B | | | | | |
| 20 Low-level testing | | | | | A | | B | | C | | | | | |

Key Areas



Levels

IQIP
98 405 SCT 7



Current situation - example

| Key Area / Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 1 Test strategy | | A | | | | | B | | | | C | | D | |
| 2 Life-cycle model | | A | | | B | | | | | | | | | |
| 3 Moment of involvement | | | A | | | | B | | | | C | | D | |
| 4 Estimating and planning | | | | A | | | | | | B | | | | |
| 5 Test specification techniques | | A | | B | | | | | | | | | | |
| 6 Static test techniques | | | | | A | | B | | | | | | | |
| 7 Metrics | | | | | A | | A | | B | | | C | D | |
| 8 Test tools | | | | | A | | | B | | | C | | | |
| 9 Test environment | | | | A | | | | B | | | | | | C |
| 10 Office environment | | | | A | | | | | | | | | | |
| 11 Commitment and motivation | | A | | | | B | | | | | | C | | |
| 12 Test functions and training | | | | A | | | B | | | | C | | | |
| 13 Scope of methodology | | | | | A | | | | | | B | | | C |
| 14 Communication | | | | A | | B | | | | | | C | | |
| 15 Reporting | | | A | | B | | C | | | | | D | | |
| 16 Defect management | | A | | | | B | | C | | | | | | |
| 17 Testware management | | | A | | | B | | | | C | | | | D |
| 18 Test process management | | A | | B | | | | | | | | C | | |
| 19 Evaluation | | | | | | A | | | B | | | | | |
| 20 Low-level testing | | | | | A | | B | | C | | | | | |

IQIP
98 405 SCT 8



Desired situation - example

| Key Area / Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 1 Test strategy | | A | | | | | B | | | | C | | D | |
| 2 Life-cycle model | | A | | | B | | | | | | | | | |
| 3 Moment of involvement | | | A | | | | B | | | | C | | D | |
| 4 Estimating and planning | | | | A | | | | | | | B | | | |
| 5 Test specification techniques | | A | | B | | | | | | | | | | |
| 6 Static test techniques | | | | | A | | B | | | | | | | |
| 7 Metrics | | | | | A | | | | B | | | C | | D |
| 8 Test tools | | | | | A | | | | B | | | C | | |
| 9 Test environment | | | | A | | | | B | | | | | | C |
| 10 Office environment | | | A | | | | | | | | | | | |
| 11 Commitment and motivation | | A | | | | B | | | | | | C | | |
| 12 Test functions and training | | | | A | | B | | | | | C | | | |
| 13 Scope of methodology | | | | | A | | | | | | B | | | C |
| 14 Communication | | | A | | B | | | | | | | C | | |
| 15 Reporting | | A | | | B | | C | | | | | D | | |
| 16 Defect management | | A | | | | B | | C | | | | | | |
| 17 Testware management | | | A | | | B | | | | C | | | | D |
| 18 Test process management | | A | | B | | | | | | | | C | | |
| 19 Evaluation | | | | | | | A | | B | | | | | |
| 20 Low-level testing | | | | | A | | B | | C | | | | | |

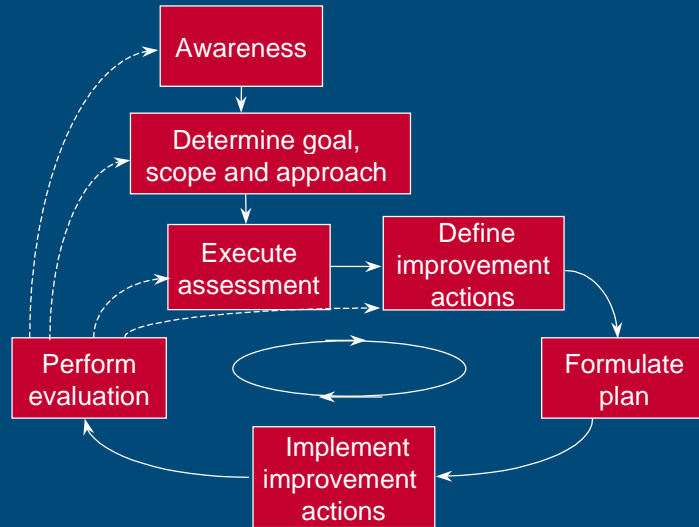


TPI Maturity Categories

| Key Area / Scale | Project | Organisation |
|---------------------------------|-------------------|------------------|
| 1 Test strategy | Controlled | Efficient |
| 2 Life-cycle model | | |
| 3 Moment of involvement | | |
| 4 Estimating and planning | | |
| 5 Test specification techniques | | |
| 6 Static test techniques | | |
| 7 Metrics | | |
| 8 Test tools | | |
| 9 Test environment | | |
| 10 Office environment | | |
| 11 Commitment and motivation | | |
| 12 Test functions and training | | |
| 13 Scope of methodology | | |
| 14 Communication | | |
| 15 Reporting | | |
| 16 Defect management | | |
| 17 Testware management | | |
| 18 Test process management | | |
| 19 Evaluation | | |
| 20 Low-level testing | | |



Process of Change



Agenda

- The TPI[®] model
- ➔ • 3 TPI Experiences



Experience 1: Large bank

- Staff department
- System and acceptance tests
- Goal
 - Cheaper & faster testing
- Assessment
 - Variety in results
 - Low maturity
- Improvement targets
 - Integration of system and acceptance testing
 - 2 year improvement period



Experience 1: Results & lessons

- Lack of “real” commitment
- Lack of central change process
- Results
 - Islands of TPI
 - Local improvements



Lessons:

- Budget
- Sponsor
- Change process in large organisations



Experience 2: Financial company

- Fast growing organisation
- Many failures in production
- Goal: better product quality
- Assessment
 - Low testing maturity
- Improvement targets
 - Control of (acceptance) test process
 - Train people
 - Create change capability



Experience 2: Results & lessons

- High visibility of internal change team
- Low visibility of external advisor
- Intensive support and training
- Results:
 - Re-assessment showed improvements
 - No other measurements
 - Test department changes manager

Lessons:

- Demonstrate improvements



Experience 3: Insurance company

- Buy instead of Make
- Test Service Centre
- Goal
 - Higher product quality
 - Shorter time-to-market
- Assessment
 - Low testing maturity
- Improvement targets
 - Better control of test process

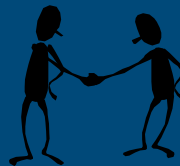


Experience 3: Results & lessons

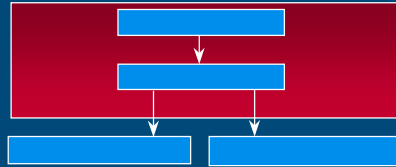
- Improvement + consolidation in stages
- First stage (9 months):
 - Improved product quality
 - Improved time-to-market

Lessons:

- Partnership
- Commitment
- Change process

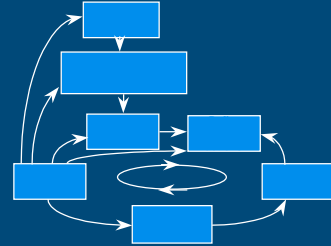


Summary



TPI-model

- Means of support
- Covers all aspects
- Based on practice
- Control
- Confidence



TPI

- Change process & organisation
- Commitment
- Demonstrate improvements



Do you really think...

Can you explain...?

Questions?

Who!

Ask now!

How?

or email: tpi@iquip.nl

website: www.iquip.nl/tpi

Where...

What??



Info about TPI:

Books: English (available);
Dutch (available);
German (Spring 2000)

Website: www.iquip.nl/tpi

Email: tpi@quip.nl



Test Management

Solutions for Project Improvement

Document: Test Management Solutions
Version: September 1999

© 1999 Prof. David Powell. All rights reserved
Any party wishing to use or reproduce text from this publication must first obtain the express permission of the author.

Contents

| | |
|---|-----------|
| 1. Foreword | 3 |
| 2. 4 main reasons for testing | 4 |
| 3. Management of Risks | 6 |
| 3.1 Risk Analysis | 6 |
| 4. Cost Management | 7 |
| 4.1 Business Focus | 7 |
| 4.2 Fault Costs | 7 |
| 4.3 Early detection..... | 7 |
| 5. Management of Time | 8 |
| 5.1 Time Squeeze | 8 |
| 5.2 Parallel Test Development | 8 |
| 5.3 Automated Test Tools | 8 |
| 6. Management of Quality | 9 |
| 7. Responsibility of the Test Manager | 10 |
| 7.1 Test Model..... | 10 |
| 8. Short Term Focus vs. Long Term Vision | 16 |
| 9. Strategic Implementation | 17 |
| 10. About Interim Technology | 18 |

1. Foreword

Organisations operate in a changing business world where they are daily confronted by new developments, new requirements, greater expectations and bigger desires on the part of their clients. Information Technology has come to play a greater role in addressing these changes. No organisation or institution can now afford to ignore IT because the commercial and operational benefits are too great. Modern day technology offers many opportunities for astute businesses.

The quality of the information flow is also becoming far more important - critical would not be too strong a word to use. The growing integration between business processes and IT means that any interruption to the normal operation of the computer systems and the business side can have disastrous effects and expensive consequences. One major US organisation recently lost its internet booking system for over 3 days, meaning anyone wanting to book with them from overseas either had to call the States by telephone or contact their rivals website. My guess is most did not call !!

The importance of prevention rather than cure of such interruptions has now become both recognised and accepted. Software testing (or Software Quality Engineering as it is called in America) has developed into an essential specialist area within the IT industry.

In practice, however, the controllability of the testing process often proves limited. While the design, development and administration of systems is usually approached very systematically, the testing process has (and in many organisations it still is) frequently conducted on an ad hoc, unstructured basis or as an after-thought.

Primarily the process has to be appropriate to the organisation in which it is to be adopted - it must fit rather than conflict with the other practices and procedures. Secondly, testing can only offer quality control if it is approached structurally and if what is produced allows for easy maintenance throughout the lifetime of that process, system or application. And finally, the use of tools and accessories is essential to any modern testing process, so as to provide greater long-term returns.

This paper looks at the need for testing to be a framework in which the areas mentioned above are used. Due to testing being a rapidly developing professional area, the structure has to allow room for new methods, services and tools to be developed.

Testing is now an essential process for any organisation and provides certain guarantees regarding the correct functioning of systems. I hope that this paper provides you with food for thought when considering the solutions for improving the testing in your organisation.

Prof. David Powell MSTI

September 1999

2. 4 main reasons for testing

There are - at least as far as this paper goes - 4 main areas or reasons why organisations undertake software and system testing.

Management of Risks

Risk Management has grown in importance within the IT industry, and I doubt I am the only person to be in software testing with a Masters degree in Risk Management.

Surprisingly many organisations still do not see that by managing the risks within both their development activities and within their testing work, they can achieve valuable long term benefits.

Management of Costs

“But the problem is that extensive testing to reduce risks costs too much”

That is a subjective attitude. It is not always apparent how much it **would** have cost had the faults not been found until after the system was in the live environment. How many orders would have been lost, how many helpdesk calls would have been logged, etc.

And costs are not always direct costs. A lost reputation might cost the organisation its very existence. If a factory or office closes, it is not just those workers who lose their jobs, but the impact on the local economy can be many times greater.

Management of Time

“Testing takes too long especially as the time to market for new products and services is becoming ever shorter.”

The increasing globalisation of markets is having a direct effect of increasing competition. And history shows us that things will not get slower and are more likely to get faster.

Products have to be brought to the market as quickly as possible. Moreover, in practice IT development projects appear to become more time consuming as the delivery date approaches.

The paper addresses ways of removing the pressure points on the testing needed, to allow the business to keep ahead of the opposition.

Extensive manual testing aimed at achieving a quality to market solution is extremely time-consuming. Automated test tools whilst offering a number of significant advantages, also have associated issues which need to be borne in mind.

Management of Quality

The quality of the application or system delivered to the market is now of crucial importance. Whilst double entries in mailing lists or address files are a nuisance, double entries in an accounting package can cost organisations their futures.

Even more serious problems can arise when modern technology throws a 'spanner in the works' of the whole business operation. Bad quality will cost you customers and will damage your reputation. In short, systems failure can have far-reaching consequences for the core business of any company or institution.

Extensive manual testing aimed at achieving a quality to market solution is extremely time-consuming.

The Test Manager

Enter the test manager.

From experience some feel they should have first train as a juggler - after all, their experience shows that will be required to keep many balls in the air - without dropping them of course - and to juggle conflicting demands.

Their mission is to help the organisations bring their products and services to the market more rapidly and to the pre-determined level of quality.

Over the next few chapters we will look at each of these areas in more detail.

3. Management of Risks

The time when testing takes place is often left to chance. It's hardly surprising then that chance plays a great part in the reliability of the IT system !

To avoid falling into this trap, testing must become a serious project within an organisation. This starts with a thorough Risk Analysis, an adequate testing budget and sufficient resources to do the work.

This may seem costly and time consuming, but the consequences are worse still.

3.1 Risk Analysis

A Risk Analysis is an absolute necessity in order to give a test manager an advanced indication or warning of where problems might be found.

In his training course on Object Orientated testing, Lee Copeland spends the first half looking at the issues, challenges and problems which OO developers face before asking the question about where should we as testers focus our test effort.

These risks or potential problems may be within the system to be tested, but could also be within the organisation, the specifications, the test environments (or lack of them), the designed tests, etc.

A person (or on big projects this might be a team of people) should be appointed with the authority to determine whether there are risks and how they should be addressed.

These risks need not simply be "known" risks. They could be issues that are identified as being a risk **should** they happen.

For example, on one project the organisation concerned had a clear desk policy, which meant that anyone could sit anywhere, and the lack of assigned desks was identified as a potential risk. As the project grew there were more people than desks, so the lack of facilities materialised as a risk. Because this had been highlighted earlier to management, a solution was already being worked out, bringing time and cost management benefits.

Risk analysis is extremely important on projects where there are more than one supplier. I worked on a project which had 3 major IT consultancys building the system plus the client doing some of the development work. All were dependent on each other but because of the rivalry and competitiveness very little communication took place between the consultancys.

If this area is taken seriously, it will assist in relieving pressure on time management, cost management and quality management.

4. Cost Management

4.1 Business Focus

In really good organisations the test managers are involved from the earliest stages of project planning and costing. This becomes even more important in consultancy projects (especially fixed price contracts) where the clients do not appreciate delays or cost overruns.

Normally at board level, an agreement for capital investment is made and will not be changed. The business is always focused on the financial aspects and good test managers should at the minimum have some understanding of the workings of the business in which they are working.

4.2 Fault Costs

The later faults are found the more money they cost to fix. In 1993 there was research done on this which produced the following information:

| Development Project Stages | Cost of fixing the fault |
|----------------------------|--------------------------|
| Operational | £100 |
| Acceptance Testing | £50 |
| Development Testing | £27:50 |
| Coding | £10 |
| Design | £4.50 |
| Requirements | £1 |

Adapted from: GILB, Software Inspection, 1993

The above chart shows that, if a fault found during the requirement stage cost £1 to fix, then it would cost 100 times that amount if the same fault was found after the system had gone live. In my experience it is likely that a fault found at the requirement stage will cost at least £50 to fix (including mantime and retesting) which means the same fault, if found after the system had gone live, would cost £5000 !!

So the earlier faults are detected then the more this will meet the business objectives of no increased costs and no delays. It also gives the test manager one other opportunity - to bring the project in under budget !!

This approach was tried on a recent project and slide 5 shows the effects that this had. There were 17 parts to the overall system, including interfaces to 3rd party software such as PeopleSoft. The project was a RAD based development with each part of the system being built in isolation, and the inherited plan was to test it in the same way.

On average 30 faults were found when going through the functional specifications (which the client had signed off !!) giving a total of 510 faults.

Due to time constraints it was agreed that these would be fixed when the detailed design documents were produced. Once these were available the same exercise mentioned above was

repeated – note we did not just re-test the fixes. A further 10 new faults were found on average

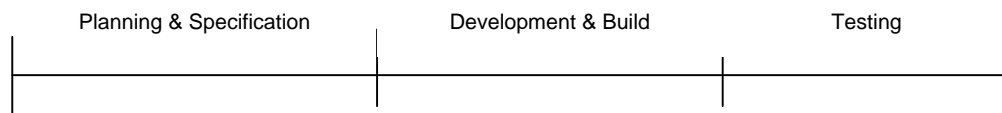
Based on the above model, if this had cost £540 to fix the functional specification faults and £765, the total would have been £1305. In reality it costs around 50 times this to fix (an

But this is insignificant when compared to the amount had they been left until system testing as is the norm with that organisation.

5. Management of Time

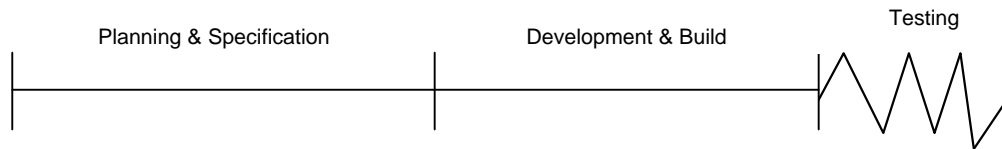
Testing must be given the attention it deserves. It is simply no longer enough to approach the examination of information systems in a 'we'll do it if we have time' way. This is hardly an improvement on the old philosophy that testing is another word for contingency !!

5.1 Test Squeeze



The above model shows at a high level the normal project phases at the start of a new project.

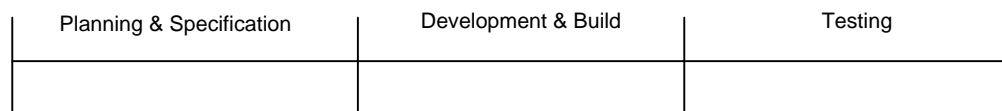
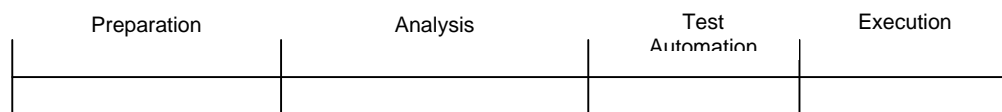
But once things start, they have the alarming habit of slipping, all except the deployment date. So the model normally ends up looking like this:



5.2 Parallel Test Projects

Parallel testing allows for the test development to be done in parallel with the system development. It also allows the documentation produced during the system development to be tested as outlined in the previous chapter. So the model would look like this:

Test Development



System Development

Obviously if the tests are to be executed manually then the Analysis phase (where the tests are designed and built) would extend and the test execution might start earlier to allow longer to run the tests needed to meet the required quality standard.

5.3 Automated Test Tool

Testing tools can speed up and simplify the testing process. The actions normally carried out by a human tester (such as mouse movements and keystrokes) are taken over by a computer program. Such actions are included in a Test Script which functions as a control program for the test tool. The Scripts can be modified, as they have an underlying programming language.

The most significant advantage of the use of testing tools is that once the Test Scripts have been recorded they can be used time and time again without any intervention on the part of the tester. For example a huge number of tests can be carried out unsupervised at night or a batch run simulated.

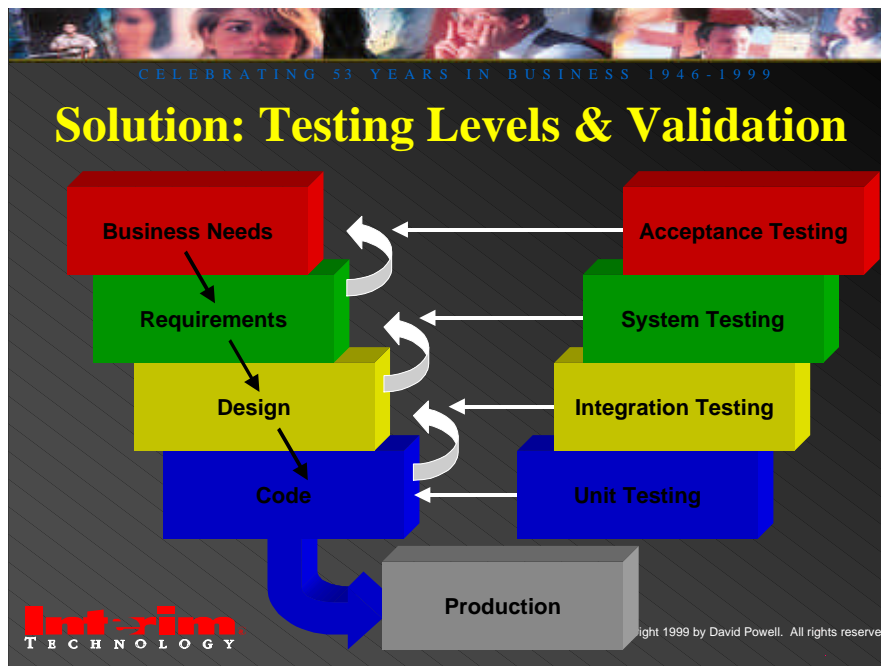
The automation of routine and often boring testing activities has the additional advantage of making the testing more reliable because the human factor is completely eliminated. There is no possibility of not being able to reproduce an error.

Maintenance of any supporting programs is an important consideration. There is little or no benefit in using these programs if they take longer than manual testing.

The use of automated is a means to an end and not the end in themselves. Test tools add speed but they do not necessarily add quality to the tests being run. After all if you automated chaos you simply get faster chaos.

6. Management of Quality

In Chapter 4, I gave the statistics of faults found on a project and the cost savings that this had on the project.



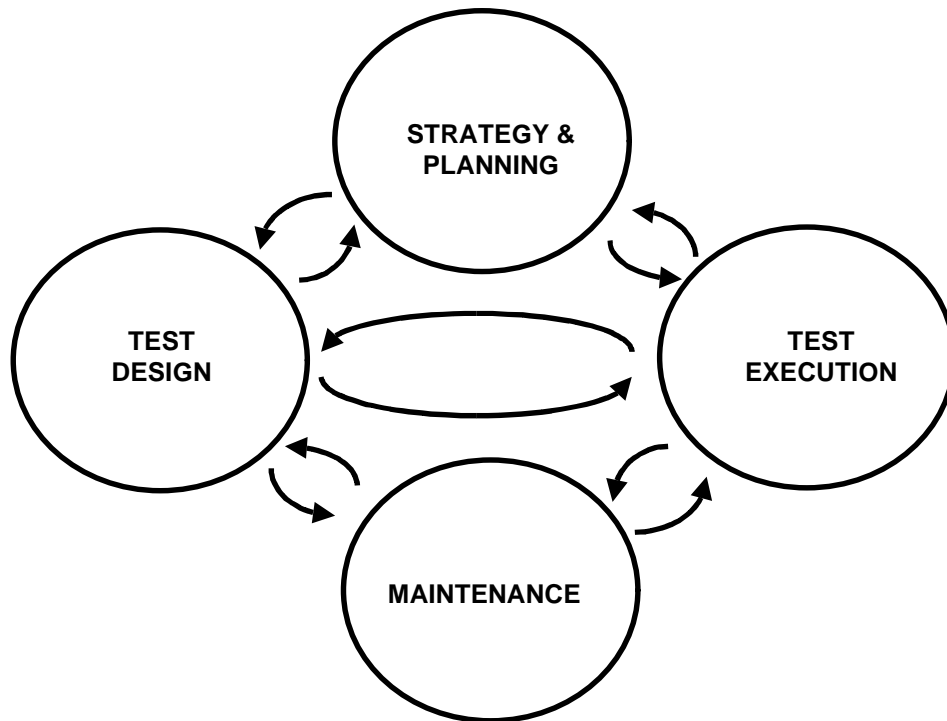
Added to this slide showing the standard V-model, are the arrows moving back up from Requirements to Business Needs; Design to Requirements; and from Code to Design. At each stage validation takes place against the former stage to make sure that what has been done is in line and to the same standards

Maintaining quality through out the design and build phase will reduce the number of faults found during System Testing, save time, and reduces costs.

But as Test Managers there should also be quality management over the tests which are produced. Good quality tests have a longer lifespan and most often can be used again and again by regression testing and then can be re-used when the system goes live for on-going maintenance.

7. Responsibility of the Test Manager

7.1 Test model



There are phases involved in setting up and implementing testing within an organisation. The model shown above does not rely on a predetermined and rigid plan. The arrows in the figure indicate that the areas need not be carried out in any set order and interact with each other.

An area can also be repeated more than once. In the case of the Strategy and Planning this should be revisited if any changes are made to the system which will be tested or changes made to the development project.

The four phases form part of each new testing project. They can also be repeated as necessary depending on the test results.

7.2 Strategy & Planning

Slide 13 outlines a series of questions which a test manager can ask at the start of the project to help them establish the basis for the Test Strategy.

The Strategy and Planning should not be a long drawn-out procedure. It is intended to:

- determine the feasibility of the project
- carry out a Risk Analysis
- define a test organisation, procedures, tasks and responsibilities
- create a clear Test Strategy for the project which should be read by all team members
- set priorities for the testing
- establish which test techniques and test tools will be used during the project
- define the infrastructure needed so that the Test Analyst can record information and the Test Executor can begin to install and use their programs, develop the automated test scripts and - of course - test the system.

The Test Strategy will form the foundation for the testing team in carrying out their activities, and will consist of:

- organisational elements;
- the testing strategy;
- acceptance criteria;
- the various testing activities;
- any products to be supplied during the course of testing, and who is to supply them.

The following quality criteria must also be included:

- the extent to which the tests will examine each of the system functions
- the way in which the test components and the implementation of the tests take place
- the manner in which the results can be carefully and controllably established.

The main problem in drawing up a timetable for testing is its dependency on the development processes of the systems. As we have already seen, any delay in the development process will almost always lead to delays in the testing process.

In formulating the planning, the following should also be considered:

- test co-ordination during the project;
- equipment for the test environment;
- structure of the test;
- test automation of all or parts of the test;
- execution of the tests;
- assessment of the test results and how these are to be logged and reported;
- the implications of any retesting required should problems be found;
- control of the test products supplied.

7.2.1 What should the testing be ?

Slide 14 outlines 3 areas which need to be addressed :

Requirements driven tests - looks that the business needs have been adequately and correctly described and ensures compliance with these requirements.

Data driven tests - checks that the system functions correctly, handles data and includes any special data cases (such as conversion mechanisms)

Error driven (or destructive) tests - looks at anticipated human activities, worst case scenarios and errors that are benign.

7.3 Test Design

At each level of test development, consideration must be given to what is going to be tested and how can we know if the test is complete ? Each test is divided into a logical structure, for example, by system area.

Next the Test Conditions are established for each of these system areas. These are then broken down into individual Test Cases which exercise the Test Conditions.

How these proceed depends on the test execution method. If the tests are to be run manually then each Test Case has Test Lines and Test Data added which will be input into the system. The very structure of this approach increases the quality of manual testing, and the tests can be produced in spreadsheets or a database.

If the tests are to be run using an automated test tool in it's standard Record and Playback mode, then the test data is recorded straight into the test tool, and the test cases saved within the software.

7.3.1 Test Conditions

Test Conditions are an important part of the overall testing process. They are a description of what needs to be tested in order to prove that the system operates according to the functional or other specifications. The Test Conditions also indicate the depth of testing required.

For example, on an client record system, the Test Conditions identified from the specifications could be that:

- a clients record can be added to the system;
- a client record can be amended to show new information;
- a client record cannot be deleted.

The expected results are normally logged at this point so that they can be compared with the actual test results to see if a fault has occurred when the tests are run.

7.3.2 Test Cases

The Test Conditions are then translated into Test Cases.

For each Test Case, the computer functions are tested to ensure that the test results comply with the expected results.

For example, the tests for the first Test Condition shown above could be:

- a client record can be added with full details
- a cleint record can be added with partial details **
- a cleint record can be added with mandatory details only
- a cleint record cannot be added without mandatory details

** There may be a number of tests run to prove different ways of adding partial details, but it is not always necessary to test every percieveable permutation unless this is part of the system specification .

7.3.3 Test Lines

The definition of Test Lines is the final phase of structuring the test preparation and contain the data to be input into the system. They can also be details of any expected messages or functions that need to be performed, such a pressing the F9 key at a certain point.

7.4 Test Execution

Again the method of execution will have a project impact. If this is done manually then there will be need for more testers within the test team. If this is done with an automated test tool then this will need people who can use these tools and are able to amend the underlying Test Script if necessary.

7.4.1 The Test Report

A clear overview of the test results, of passes or any problems that have come to are essential in order to be able to deal with faults. Any carelessness can undermine the quality to market of the application.

With automated test tools they automatically produce a Test Report in which changes in the expected result are shown. These may be minor (such as a wrong screen message) but could be more serious (such as a system which cannot cope with a certain number of transactions)

With manual testing there needs to be mechanism put in place to log the test results and to log any faults found during the running of the test.

7.4.2 Results administration

If an error requiring attention is discovered in the system, it is important that the procedure for dealing with this is carefully monitored. The administration should describe and follow through all errors. Everyone involved in systems development or maintenance should be informed of the errors or problems in the relevant version.

Faults and errors can be divided into various categories, depending on the gravity of the problem. The most important category is one where the implementation of the system becomes impossible. The second category can include those problems which affect the proper functioning of major parts of the system. A third category fault may be for the more cosmetic problems, such as spelling mistakes in the help file, or an incorrect error message.

This division into categories is useful for any management report because it clearly indicates the nature of the problems. By regularly supplying such overviews, you will be able to closely monitor how the quality of the system is developing.

Repetition of some tests may be essential if major errors requiring attention are discovered in the system, and periodic full regression tests are also advisable, especially on RAD projects.

Once the tests show the required results, you can comfortably move onto the next phase.

7.5 Maintenance

Test maintenance can be roughly divided into four components:

- setting up and maintaining the Test Reports
- managing the results of the tests
- managing the different versions of the system being tested
- transfer of the test products to the production/live environment.

8. Short Term Focus vs. Long Term Vision

8.1 Re-usable testing products

Testing products are often subject to poor reuse. Many organisations find that they have to set up tests from scratch each time there is a minor modification to the system. Frequently this is due to negligence.

The requirement for a time management means that tests produced during the development phase of a system must be easily maintained and totally reusable.

On the other hand the quality management demands that the various stages of testing can produce an accurate risk assessment.

An easily maintainable test suite should be available from the very outset - one which can be used time and time again with just minor modifications when necessary.

The time savings will become particularly noticeable when updates involve only five to ten percent of the system. In testing the other 90 – 95% will be retested using existing test material.

Because the various testing resources are used time and again, the initial investment pays for itself after a few repetitions.

9. Strategic Implementation

In conclusion, the only way to address the four key areas mentioned in Chapter 2 (time, quality, cost and risk management) is to have a strategic corporate solution to meet all the organisations testing needs.

One organisation I met had 40 different development project teams who each focused on one particular system which was used within that company. Each project manager defined how their development work was to be tested, giving a possible 40 different ways of testing. Once they were happy with the system it went live and the tests passed to the maintenance team.

How much easier, cheaper and faster it will be for the organisation when they adopt a strategic test approach, method and standards across all 40 projects.

A reusable testsuite reduces the costs and time to market for upgrades and new functionality, and I would suggest quality to market will be dramatically increased because the risks are reduced.

10. About Interim Technology

Software Quality Management

With today's focus on achieving more with fewer resources, applying consistent quality to the software engineering process is critical. The costs of inadequate quality management include lost opportunities, excessive corrective procedures, reduced competitiveness, and reduced profits.

Interim Technology, The Consulting Group has developed a range of services and products designed to help Information Technology departments deliver defect-free software on time and within budget. Interim's quality management procedures, based upon international standards, are compatible with any software development life cycle and have been proven in the field on hundreds of projects, on four continents, for more than twenty-five years.

Quality Procedures Development and Implementation

Interim has refined its extensive experience into practical, easy-to-implement adaptable strategies and procedures. Interim can help integrate these or similar approaches with organizational philosophies, methods, and goals.

Quality Management Handbook (QMH)

A guide to Interim's comprehensive Quality System and its applicability to all aspects of software development and maintenance. Based on international standards, the QMH is designed to be customised for in-house work and that provided by vendors.

Project Management Handbook (PMH)

The PMH contains steps and guidelines for effective management and control of software projects. The PMH is compatible with all recognized life cycle methodologies and project management tools, helping businesses get the most out of their software development process.

VALI/TEST Pro (SM)

Long recognized as the most comprehensive approach to the testing of custom developed or package-based software, VALI/TEST Pro covers all phases of the software life cycle. It is a Windows™-based hypertext tool, which is also available in book form.

Software Testing and Validation

Testing is vital in the creation of defect-free software. Effective testing not only identifies design and coding errors, but validates that the software truly meets business needs. Interim Technology's validation approach focuses on requirements-based testing through the complete life cycle, including unit, integration, acceptance and post-implementation maintenance testing. VALI/TEST Pro, is the foundation of a range of testing services. This online methodology tool provides the most comprehensive approach to software testing and is the foundation of a range of testing services including: testing process assessment, testing, testing strategy and planning, user acceptance testing, testing process implementation, testing for the year 2000, and test project management.

Quality and Productivity Metrics

Interim's staff of professionals will deliver a metrics strategy planning to help identify the software metrics most closely aligned to an organization's mission and goals. Metrics program implementation assists with the planning and implementation of a Metrics program.

Quality Services for Project Management

Interim helps organisations apply the fundamentals for success contained in our Project Management Handbook (PMH) and Quality Management Handbook (QMH). Doing so helps lay the foundation for

project success and forms the basis of a number of Interim's project management services that include: request for proposal (RFP) development and review, project office, contract management, and independent quality assurance.

Education and Training

Implementing effective quality management requires education and training at all levels, in addition to long-term management commitment. Interim Technology offers a full range of educational programs that can be customized to fit specific needs, as well as for individual mentoring and coaching. All are designed to instill and reinforce the understanding of quality principles. Interim's programs address both management and implementation issues alike.

User Services

The goal of User Services is to forge a partnership between business and MIS management to ensure dependable systems that meet business requirements. Interim's user services staff rely on a proven process that integrates user-related elements from several key disciplines to ensure maximum productivity. This specialty can provide support in the following areas: training, custom application training, requirements definition, acceptance testing, user project management, project office, package selection and implementation, and user documentation.

Quality Consulting

A proven project initiation and management process is essential in order to ensure dependable, high quality results with a minimum of inefficiency, risk and cost. Interim can assist by applying the fundamentals for success contained within its Project Management Handbook (PMH) and Quality Management Handbook (QMH).

TEST/CYCLE®

Interim Technology, The Consulting Group's software testing specialty offers a comprehensive set of services and products, all based on the VALI/TEST Pro (SM) approach to testing, that can be tailored to an organization's testing objectives.

TEST/CYCLE® , an advanced PC- or LAN-based software automates management of the entire testing process. TEST/CYCLE complements and enhances the software development process, whether it includes a traditional life cycle, client/server, GUI, RAD, or other methods. It thoroughly supports all testing levels, including unit, integration, acceptance and regression testing.



CELEBRATING 53 YEARS IN BUSINESS 1946-1999

Test Management - Solutions for Project Improvement

November 4, 1999

**David Powell, MSTI
Management Consultant
Software Quality Management**



© Copyright 1999 by David Powell. All rights reserved.

CELEBRATING 53 YEARS IN BUSINESS 1946-1999

Why Test ?

- Management of Risks
- Management of Costs
- Management of Time
- Management of Quality
- Test Management



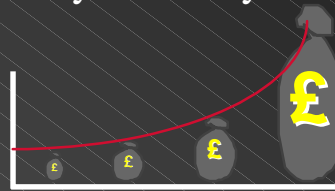
© Copyright 1999 by David Powell. All rights reserved.

Risk Management

- Faults are risks until they are found

BOMB tick TICK **TICK**

- Finding them later will cost you money



© Copyright 1999 by David Powell. All rights reserved

Cost Management

.... Defect Repair @ Different Project Stages



Adapted from GILB, Software Inspection, 1993

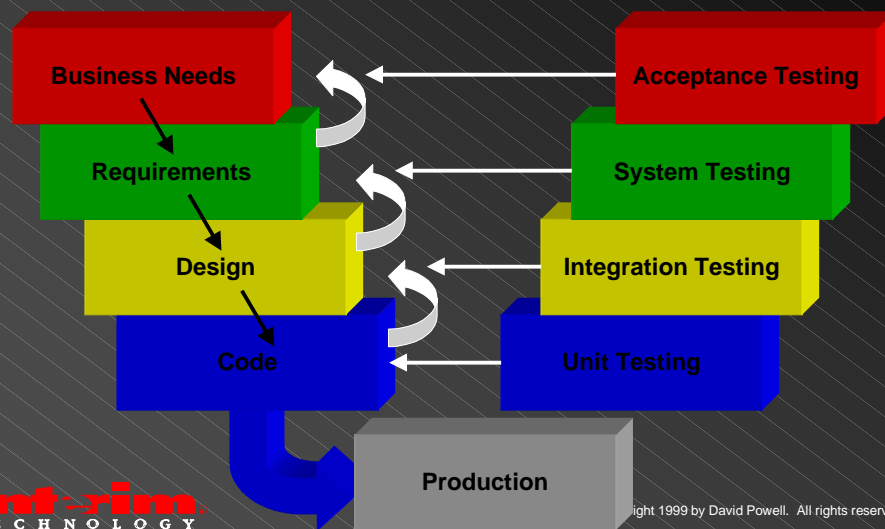


© Copyright 1999 by David Powell. All rights reserved

The 17 x 30 Rule

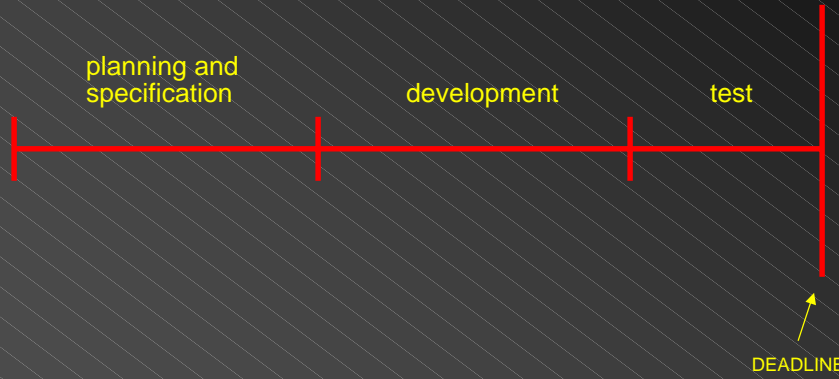
- **Functional Specifications**
 - New International System for Medical Insurer
 - 17 System areas being build in isolation
 - Average 30 faults found in each of the 17 areas
 - Total : 510
- **Detailed Design Documents**
 - Produced for each of the areas to corret the faults
 - Average 10 faults found in each of the 17 areas
 - Total : 170
- **& the business signed off all these documents !!!**

Solution: Testing Levels & Validation



Time Management

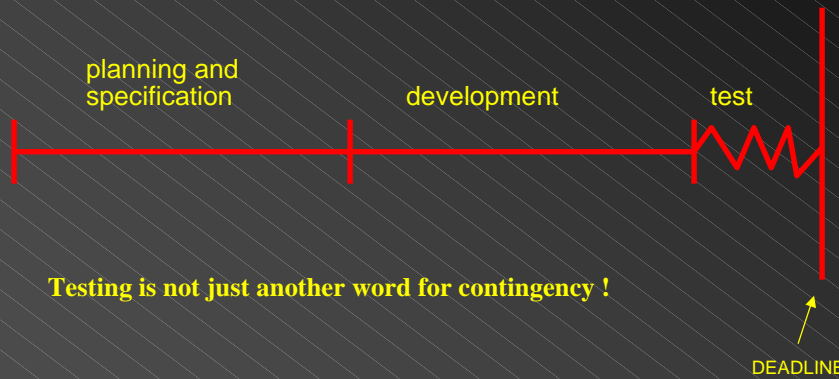
..... testing is often put under pressure



© Copyright 1999 by David Powell. All rights reserved.

Time Management

..... testing is often put under pressure

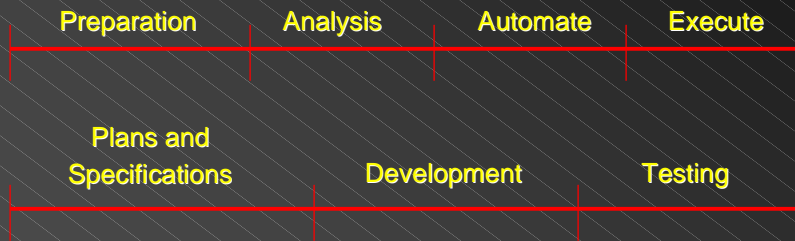


© Copyright 1999 by David Powell. All rights reserved.

CELEBRATING 53 YEARS IN BUSINESS 1946-1999

Solution: Parallel Development

SQM Development



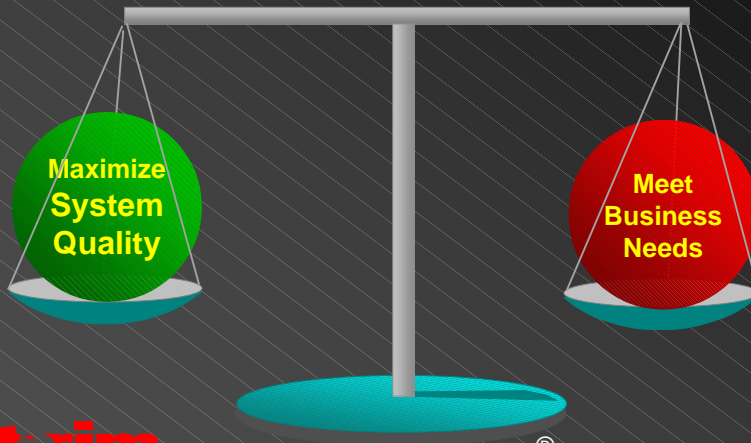
System development



© Copyright 1999 by David Powell. All rights reserved

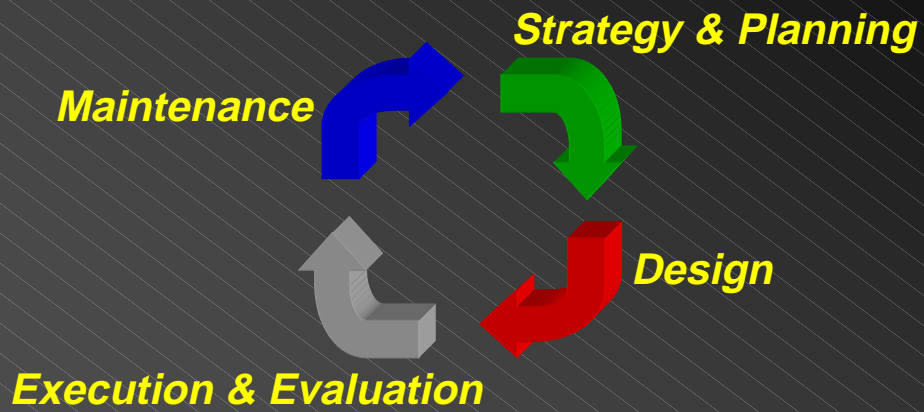
CELEBRATING 53 YEARS IN BUSINESS 1946-1999

Management Objectives



© Copyright 1999 by David Powell. All rights reserved

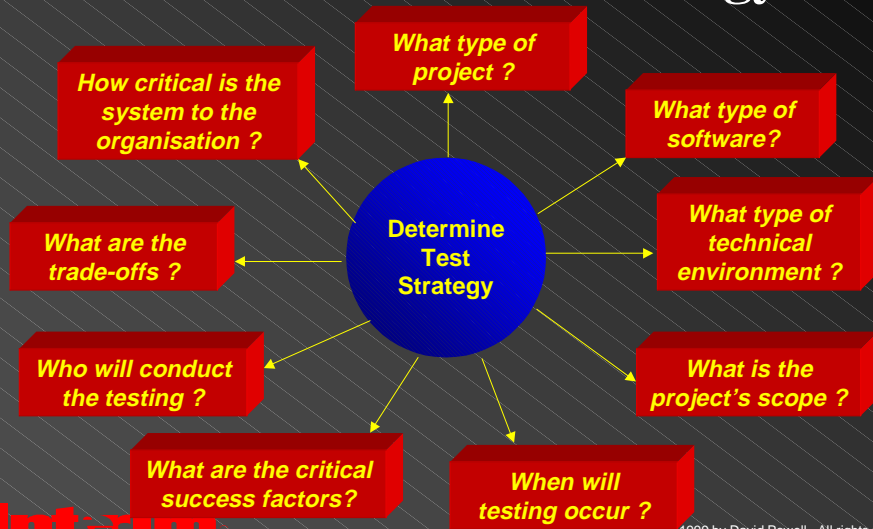
Project Activities



Common Activities

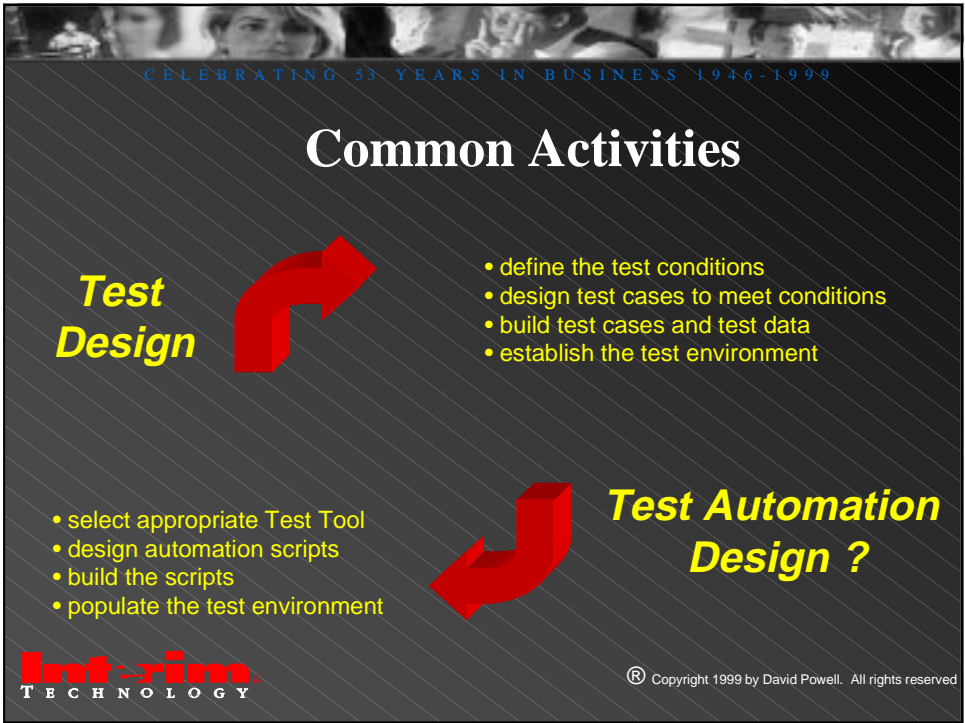
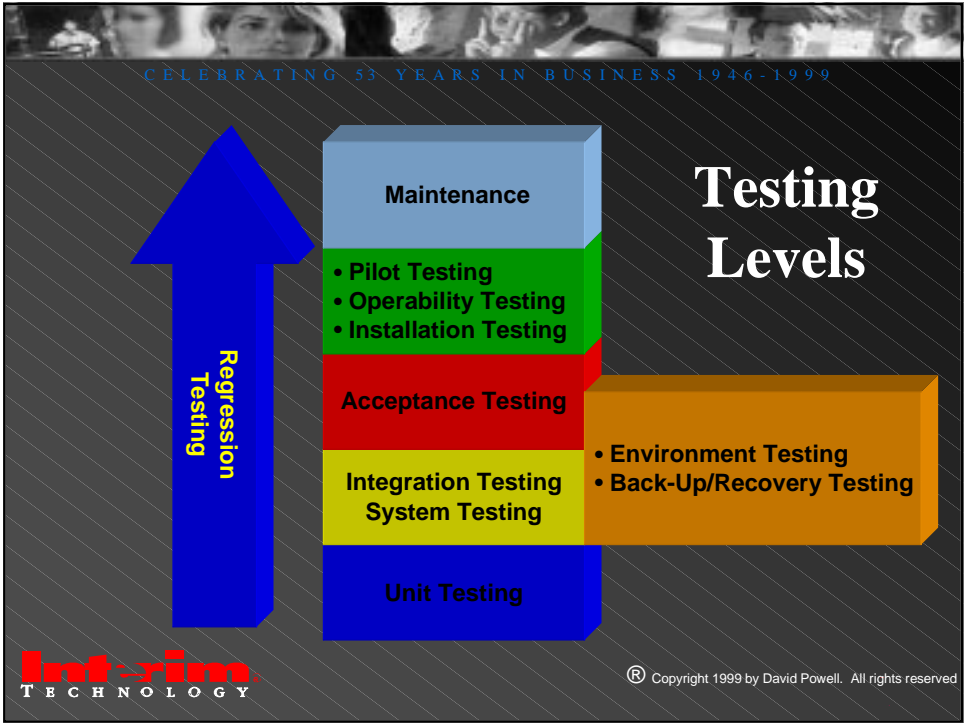
- ### Strategy & Planning
- establish the test strategy
 - investigate the project feasibility
 - identify risks and problems
 - establish test organisation and procedures
 - plan test environment
- 

Determine the Strategy



Testing Should Be ...

- **Requirements driven**
 - Ensuring the business needs have been correctly described
 - Ensuring compliance to the requirements
- **Data driven**
 - System functions correctly, handles data, including special data cases
 - Data has integrity
 - Data is safe (backup & recovery)
- **Error driven - Destructive Testing**
 - Anticipate human activities
 - Errors are benign
 - Worst-case



Common Activities



Execution & Evaluation

- execute tests
- report faults
- retest fix releases
- produce reports
- validate reports



© Copyright 1999 by David Powell. All rights reserved.

Common Activities



Maintenance

- on-going keeper of the tests
- maintain test cases
- maintain automation scripts

Well defined tests can be used daily if the maintenance requires this



© Copyright 1999 by David Powell. All rights reserved.

Plan for Maintenance

- **Prepare for Maintenance during Development**
- **Keep Deliverables and Documentation up-to-date**
 - Requirements; Test Design; Test Cases; Test Data; Test Results
- **Maintain all related deliverables during error correction**
 - Requirement specifications
 - Program specifications
 - Design specifications
 - Defect reports

**Good Maintenance provides huge Time, Cost,
Quality and Risk Management Rewards**



© Copyright 1999 by David Powell. All rights reserved.



Any Questions ??

Future Contact

David Powell MSTI
sqm-uk@interim.com



© Copyright 1999 by David Powell. All rights reserved.

1. MANAGING YOUR TEST COSTS

This abstract is meant to be a short introduction to the presentation ‘Managing Your Test Costs’. The accompanying text written by Jens Pas and Mario Jansen will give more detailed information regarding this issue.

| | |
|--------------------------|---|
| Title | Managing your test costs |
| Bullets | <ul style="list-style-type: none">• How to forecast your testing budget• How to manage your testing cost• How to optimise your testing cost |
| Intended audience | <ul style="list-style-type: none">• Test and Business Managers• Advanced testing professionals |
| Type | Paper |
| Paper | Jens Pas, Operations Manager @ ps_testware |

2. PREFACE

This document briefly explains the cost issue of testing as discussed in the tutorial “Managing Your Test Costs”.

The tutorial covers the issue of estimating your test budget, managing your costs while you are testing and finally addresses how to optimise your organisation in order to reduce the costs associated with testing.

The tutorial uses real life data to illustrate the issues discussed. Academic cost models that are, however, applicable in practice are presented. They will allow the attendee to structure and manage his or her cost issues. The tutorial provides a case that the attendees will solve during the tutorial.

3. INTRODUCTION

When discussing the testing of software, the issue of cost rapidly comes to surface. Responsible for this hyper-sensitivity for cost, is the fact that many people consider testing as a necessary evil, that has to be kept as small as possible. Testing is not making a contribution, testing is a pure remedial activity that helps to obtain the quality that we should have programmed in the first place. At least, this is what is said. This tutorial will not discuss why the above statements are incorrect, nor why testing does provide a great deal of contribution.

We start the tutorial from the axiom that **we must test**, whether we like it or not. Consequently, we must consider the costs associated with the testing activity. Three questions pop up when test costs are discussed:

- how to forecast the testing budget;
- how to manage the costs while you are testing;
- how to optimise the testing costs.

These questions will be discussed in more depth in the following chapters.

4. HOW TO FORECAST THE TESTING BUDGET?

According to the Gartner Group about 50% of all software engineering effort is spent on verification and validation. The other 50% concerns the building activities (e.g. analysis, designing, programming, documenting,...). If one knows his or her development budget, one can simply determine the testing effort. The problem with this way of defining the test budget is that it includes all the "test" work done by the builders (analysts, designers, and programmers...). As such, it is difficult to distinguish the effort done by professional independent testers.

Like the concept "cost", testing is also a meaningless word. Meaningless since it tries to cover to many activities. In order to measure the testing effort properly, one first has to distinguish the various activities involved in testing. We define eight types of effort done when testing. Consequently, they are eight budgets to forecast:

| | |
|------------------|---|
| Defect tracking | The time spent on the management and follow-up of the registered defects, including but not limited, to controlling whether repaired defects are waiting for closing and the creation of defect reports. |
| Test development | The time spent on the development of the test procedures and test cases, including but not limited to the prologue and epilogue procedures, the scripts, the test-the-test activities and the initialisation of the test bed. |
| Test execution | The time spent on the execution of the tests with the intent of finding errors in the product that is tested, including but not limited, to the new tests and regression tests (regression tests as a consequence of the interim versions of the product that is tested). |
| Test maintenance | The time spent on the adjusting and updating of the test ware (scripts, test cases,...), necessary for the different versions of the product under test. |
| Test overhead | The time spent on activities supporting the test process, including but not limited to the communication through meetings or reports concerning appointments with regard to the test organisation, escalation of problems, the creation of administrative reports for invoicing, giving or requesting support regarding test knowledge. |
| Test planning | The time spent on the creation of the Test Plan, including but not limited to the writing of the document, the creation of the Test Requirements Hierarchy, the meetings, the analysis of the Test Bed, the making of the planning. |
| Test preparation | The time spent on the creation of the Test Assignment (sort of Test Project Plan), including but not limited to the writing of the document, the necessary interviews and meetings, the analysis of the test basis and the installation of the test organisation. |
| Test repair | The time spent on the correction of the testware (scripts, test cases,...). This activity is mostly started as a consequence of the registration of a defect, after which the test and not the product turns out to be wrong. The time does not concern the test activities during the development of a test. |

Base ratios as experienced by ps_testware¹:

¹ These figures are derived from new development projects in a Client/Server environment, using a RAD-development approach, a automated test tool with defect tracker and Requirements Management tool. It must be noted that these figures can strongly vary if the previously mentioned elements change. The use of tools (both for automation and for management) will heavily influence the activity budget distribution.

| | |
|----------------------------|--------|
| Defect tracking | 10-15% |
| Test development | 25-50% |
| Test execution | 1-15% |
| Test maintenance | 0-10% |
| Test overhead | 25-30% |
| Test planning | 10-15% |
| Test preparation | 10-15% |
| Test repair | 0-10% |
| <hr/> Total testing budget | 100% |

Correctly estimating the required time to complete a task involves proper project management skills. It is striking how many people over-estimate the probability of their estimation. Most project managers or leaders rate their abilities to estimate time near 80%. It can easily be mathematically proven that even the best project leaders, using a “classical behaviour” cannot score higher than 30% to 40%. This means that every estimation is over optimistic and as such is prone to overrun.

Further issues have to be taken into account, when estimating time are the software engineering maturity of the organisation and its people and the intrinsic quality and complexity of the software that is written (based upon historical facts).

5. HOW TO MANAGE THE COSTS WHILE YOU ARE TESTING?

Once the testing has started, the test manager or project manager/leader must make sure that the testing is effective and efficient. In other words, the testing must be done with the highest return. This means that the manager must focus on the contribution generated by the tester, rather than on the volume of testing (testing hours, code coverage, etc.). Furthermore, we may not forget that the manager may not overrun his testing budget.

If the manager wants to achieve the highest contribution, he must know the constraint that prevents him from an even higher return. The test manager will seek for the bottleneck that does not allow him to make even more money (read: software quality). Again we will apply Theory of Constraints to manage this issue. This time we will not focus on time but on defects or bugs.

Consider the following process. A developer makes a piece of software. When finished, a tester is asked to test it. The fact that a tester is added to process (much too late, but that is another more methodological matter) confirms that the provided software most probably contains errors. So, the tester receives software of an “unknown” quality. He has been added to the process to help increase the software quality, by finding errors and having them repaired or by confirming that there where no errors. In the latter case, which is quite hypothetical since there are always errors to be found, he will not have increased the quality but he will have reduced the risk “taking into production”. In short, the tester has to find defects. In order to increase the quality of the software, the defects must be repaired. When a defect has been repaired the tester has contributed to the quality increase. It must be noted, however, that the repair was executed by the developer and not by the tester. Consequently, the tester’s contribution is dependent on the repair work of the developer. From a management control point-of-view, this is a complex issue that requires particular attention.

Two facts from this story have to be kept in mind:

1. Testers provide contribution when they find errors, which are repaired
2. Developers, and not the testers, do the repairing.

The way to manage the test process consists as such out of the management of the flow of bugs. The test manager must focus on the amount of bugs that are found in the database and the status they have. Many bugs with the “Newly found” status indicate that the tester does a good job in finding errors, but that there is no one looking at them in order to get them repaired. Hence, no contribution will be generated. This situation looks familiar, doesn’t it? How many times didn’t a test manager add a tester to the project, thinking that he would illustrate that all was done well. No repair time was foreseen. The tester was asked to come and test the last month before “going live”, regardless of his findings. As such, no resources were planned to look at the defects found and to see whether they need repairing.

Managing the test process through the flow of defects is called Throughput Management. It is a method to optimise the throughput of solved defects through the system. Here a particular metric was defined called “Defect Throughput”.

“The Defect Throughput is the pace at which the system (the project team) produces solved defects.”

By managing the Defect Throughput a minimal test cost will be made. This method relates to the Throughput Accounting² method used in Cost Accounting when establishing optimal product mixes in product manufacturing processes in factories.

6. HOW TO OPTIMISE THE TESTING COST?

Managing according to the Defect Throughput is good for optimising and controlling the operational test activities, it does not, however, help to improve and optimise the test costs in the long run. In order to reduce the testing costs, managers must identify what these costs are and what activities are driving them. Here we enter the field of Activity Based Costing (ABC) or Activity Based Management (ABM). With ABC, costs are seen as the consequence of one or more activities. Rather than to simply eliminate steps in the process in order to cut costs (e.g. reduce test time or test resources), the relation between an activity and the created costs is determined. The relation is called the “cost driver”. For instance, the cost of setting up quality test scripts depends on the “testability” of the product and of the means provided to test the product. An application that was developed in a very exotic programming language or environment will have a hard time getting fully tested at an acceptable cost using standard testing tools. Not using tools might result in poor and inefficient test quality, developing proprietary tools might also induce more unwanted costs. In the latter case, the test-tool must also be tested before taken into operation.

Using an ABC-system will facilitate the cost reduction objectives of continuous improvement³. It does this by allowing managers to gauge to the cost consequences of decisions to change the performance or use of activities. Activities may be reduced, eliminated, substituted by alternative solutions or shared, resulting in benefits through economies of scale.

It is here that issues such as the “Early-defect-discovery” must be situated. It is theoretically correct that the sooner a defect is found, the less costs it creates. In practice, however, we are confronted with the late discovery of sometimes simple and obvious bugs. Here the power of the V-model, as described by Glenford Myers⁴ is too limited to provide an implementation scheme to really find those bugs early in the process. Using the V-model as framework on one side and applying ABC as a driver on the other, a well-structured optimisation might be achieved in every organisation. Applying ABC will drive towards an optimal software engineering process, taking into account proper analysis, documentation, verifications, designs, development and validations.

² Ruhl, Jack M., Journal of Cost Management, “The Theory of Constraints within a Cost Management Framework”, pp.16-24, November/December 1997

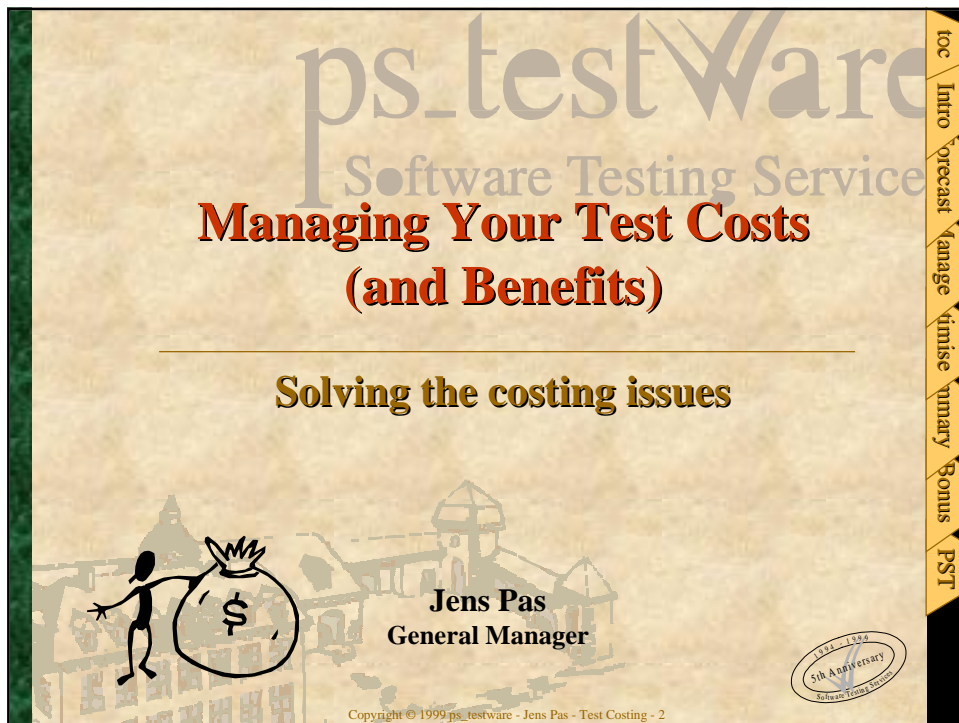
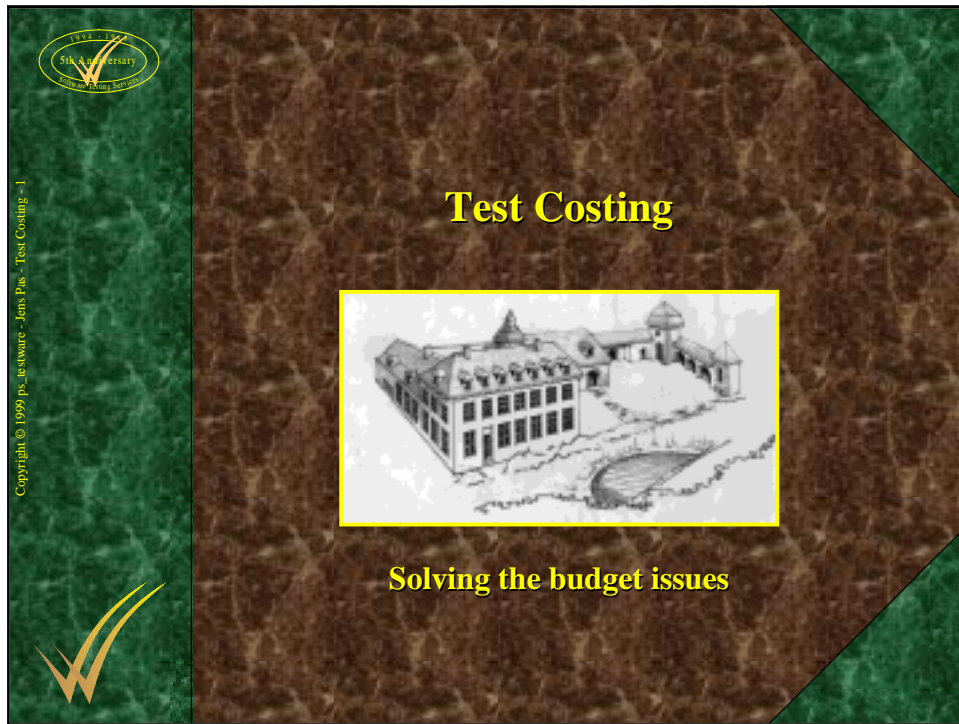
³ Turney, Peter B.B., Cost Management, “How Activity-Based Costing helps reduce Costs”, pp. 29-35, Winter 1991.

⁴ Myers, Glenford, *The Art of Software Testing*, John Wiley and Sons, 1979

Combining the optimisation of the test cost with forecasting test cost leads us to another concept of Management Accounting, Target Costing. With Target Costing we define the maximum cost we may incur during the development process. The manager must apply all possible creativity to provide the required quality, given the maximum target cost. Target Costing, though until now little documented, will probably be the answer to real Rapid Application Development (RAD) projects. Their development targets are already well institutionalised.

7. THE COST OF THE UNFOUND DEFECT

What has not been discussed is the cost of the unfound defect. This article has focussed on the costs associated with finding bugs. Some bugs, however, are not found and create damage once the software is applied in reality. It is unfair to call this the cost of the bug or even the unfound bug and not to allocate it anywhere. It is more correct to attribute this incurred cost to the development of the application and to add this to the development budget of the project. By labelling this cost as such, the sensitivity of the project leader towards the quality of his product will increase significantly. Since he has a budget responsibility, putting bugs in his budget is the same as making him explicitly accountable for the quality of the software. This is after all what we want if we hope to manage and improve the software engineering process.




Agenda


- **Introduction & Background**
- **The costing issues**
- **Forecasting your test cost**
- **Managing your test cost**
- **Optimising your test cost**
- **Questions**

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 3



toc
Intro
Forecast
Manage
Optimise
Summary
Bonus
PST





ps_testWare
Software Testing Service

Introduction & Background


Why this Topic?

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 4



1994 - 1999
5th Anniversary
Software Testing Service



toc
Intro
Forecast
Manage
Optimise
Summary
Bonus
PST

Why do we test ?

To improve the *profit* of your company!

$$\text{Price} - \text{Cost} = \text{Profit}$$

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 5

toc Intro Forecast Manage Analyse Summary Bonus PST

(How much) do we have to test ?

Man-hours testing
Test ware (tools, infra)
Test Management
Repair costs
Cost of ownership
Opportunity cost

Test ware assets
Repair costs
More rework
Quality knowledge

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 6

toc Intro Forecast Manage Analyse Summary Bonus PST

Facts

- **Axiom:**

We have to test.

- **Testing is the necessary evil**
- **Testing is remedial**
- **Testing does not provide contribution**

We have to make test costs.

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 7

ps_testware
Software Testing Service

toc Intro Forecast Manage Minimise Summary Bonus PST


The test cost issues

- **Three questions:**
 - How can I forecast the cost of structured testing?
 - How can I manage these costs during the execution of a project?
 - How can I optimise test costs in the future?

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 8


ps_testware
Software Testing Service

toc Intro Forecast Manage Minimise Summary Bonus PST




Forecasting test costs

Development strategy
Base ratios




Copyright © 1999 ps_testware - Jens Pas - Test Costing - 9




toc Intro Forecast Manage Minimise Summary Bonus PST

Forecasting test costs

- **Know your current test efforts?**
- **Assess your organisation on “structured test readiness”**
- **Use a forecasting model**



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 10



toc Intro Forecast Manage Minimise Summary Bonus PST

Know your current test efforts

| Name | Devel | Sys/AT | Oper | Effort | Cost(K) |
|------------------|--------------|--------------|-------------|--------|---------|
| Developer 1 | 20-40 | 19 | 18-24 | 57-83 | 21-30 |
| Developer 2 | 12-18 | 1-2 | | | |
| Developer 3 | | | | | |
| Developer 4 | | | | | |
| Sub-Total | 32-58 | 20-21 | 18-2 | | |
| Tester 1 | 10-12 | 33 | | | |
| Tester 2 | | | | | |
| Sub-Total | 10-12 | 33 | | | |
| User 1 | | 8 | 2 | | |
| User 2 | | | | | |
| Sub-Total | | 8 | 2 | | |
| Other 1 | | | | | |
| Sub-Total | | | | | |
| Totals | 42-60 | 61-62 | 20- | | |
| Pcts | 23% | 28% | 10 | | |

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 11

Assess your “structured test readiness”

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 12

A forecasting model (1/2)

- **Step 1: Estimate the development budgets**
 - Development strategy
 - Historical figures

| Development Budgets | % of total budget |
|------------------------------------|-------------------|
| Building & Verification | 40% - 60% |
| Module & Integration Testing | 5% - 20% |
| System Testing | 20% - 30% |
| Babysitting (& Acceptance Testing) | 5% - 10% |
| Overhead (Test Mgt.) | 5% - 10% |
| Total | 100% |

Source: ps_testware project database

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 13

toc
Intro
Forecast
Manage
Finalise
Summary
Bonus
PST

A forecasting model (2/2)


- **Step 2: Distribute budget on test activities**
 - Base ratios

| Test Activity | % of total budget |
|----------------------|-------------------|
| Test Preparation | 5% - 10% |
| Test Planning | 5% - 15% |
| Test Bed set-up | ≤ 5% !?? |
| Test Development | 25% - 50% |
| Test Execution | 10% - 30% |
| Test Repair | ≤ 5% |
| Test Maintenance | ≤ 10% |
| Defect Tracking | ≤ 7% |
| Test Report | ≤ 5% |
| Overhead (Test Mgt.) | 5% - 20% |

Source: ps_testware project database



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 14

toc
Intro
Forecast
Manage
Finalise
Summary
Bonus
PST



Managing Test Costs

Theory of Constraints



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 15


toc Intro Forecast Manage Optimize Summary Bonus PST

Managing test costs

- **Classic metrics**
 - number of defects
 - number of hours of testing
 - logic coverage

Treat the test (and development) process as a production process

Application of Theory of Constraints (TOC)



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 16

toc Intro Forecast Manage Optimize Summary Bonus PST

Intermezzo: Theory of Constraints

How many cars can be made per hour ?

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 17

toc Intro Forecast Manage Analyse Summary Bonus PST

Intermezzo: Theory of Constraints

- **Define the units to measure contribution!**
- **1. IDENTIFY** the constraints
- **2. Decide** how to **EXPLOIT** the constraints
- **3. SUBORDINATE** everything else to the above-mentioned decision
- **4. ELEVATE** the constraints
- **5. Be aware of INERTIA!** If the constraint has been broken, go back to step 1.

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 18

toc Intro Forecast Manage Analyse Summary Bonus PST

Intermezzo: Theory of Constraints

R1
(10units/h)
Testing

R2
(7 units/h)
Repairing

R3
(14 units/h)
Re-testing

What are the units ?

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 19

toc Intro Forecast Manage Analyse Summarise Summary Bonus PST

What is the testing process?

- What is testing ?

Testing is the process of executing a program with the intent of finding errors

Source: Glenford Myers (1979)

- Why ?

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 20

toc Intro Forecast Manage Analyse Summarise Summary Bonus PST

What are the units of the process?

- Goal of testing

Improve the quality of the product under test

- Quality will improve if we find defects and repair them
- So, the test process produces solved defects!
- Solved defects \approx Passed requirements

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 21

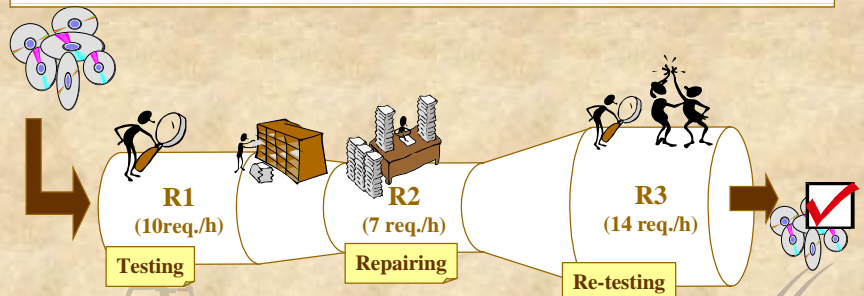
ps_testware
Software Testing Service

toc
Intro
Forecast
Manage
Analyze
Summary
Bonus
PST

Main metric

- Requirements Throughput

The pace at which the system produces passed requirements.



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 22

ps_testware
Software Testing Service

toc
Intro
Forecast
Manage
Analyze
Summary
Bonus
PST

How to manage in practice

- Register all your “test” requirements in a **Test Repository**
- Register all defects in this repository and link to the related requirement
- Follow-up status of defects in the repository
- Use the repository to report the **Requirement Throughput per “activity”**

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 23

toc Intro Forecast Manage Analyse Summary Bonus PST

Pitfall!!

- Focus of the tester

Find defects

- Focus of the test manager

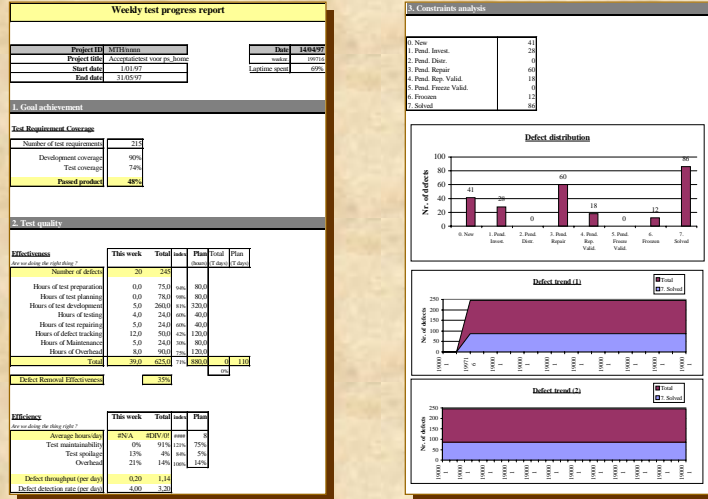
Requirements Throughput

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 24

toc Intro Forecast Manage Analyse Summary Bonus PST

Example: Progress Report



Microsoft Excel + SQA TeamTest

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 25

Optimising Your Test Costs

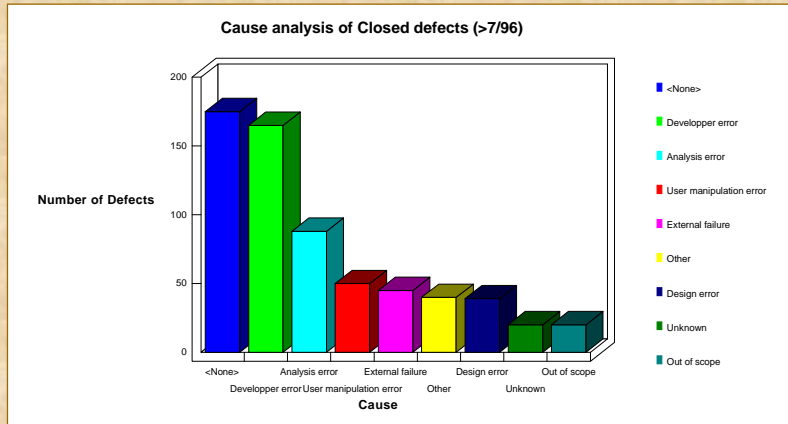
Learn from defects
Activity Based Costing

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 26



Learn from defects

- Causal analysis (pareto) of solved defects

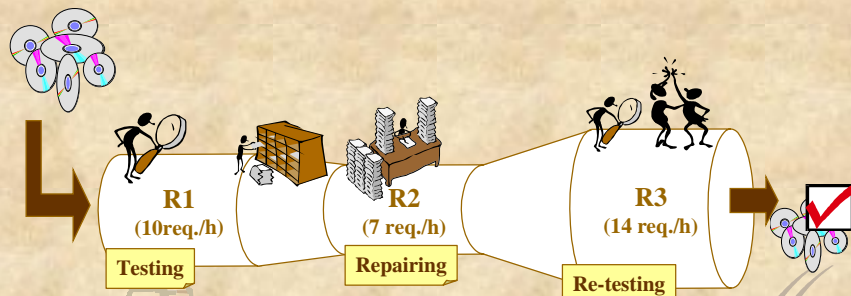


Copyright © 1999 ps_testware - Jens Pas - Test Costing - 27

Optimising the test costs

- Optimise the test process (ABC)

- Define activity pools
- Look for cost drivers



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 28

Optimise the test process

- **Activity pools**
 - Test Preparation
 - Test Planning
 - Test Bed set-up
 - Test Development
 - Test Execution
 - Test Maintenance
 - Test Repair
 - Defect Tracking
 - Test Reporting
 - Overhead

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 29

toc Intro Forecast Manage Optimise Summary Bonus PST

Test Preparation

- **Number of test levels (V-model)**
- **Number of sub-systems**
- **Number of dependencies and project constraints**

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 30

toc Intro Forecast Manage Optimise Summary Bonus PST

Test Planning

- **Number of user/test requirements**
- **Availability and quality of the test basis**
- **Business knowledge of the tester**
- **Complexity of the test bed**
- **Amount of concurrent testing**
- **Resource constraints (human and infrastructure)**

toc Intro Forecast Manage Minimise Summary Bonus PST



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 31

ps_testware
Software Testing Service

Test Bed set-up

- **Complexity of the software architecture**
- **Nr. of infrastructural requirements**
- **(see also test planning)**

toc Intro Forecast Manage Minimise Summary Bonus PST



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 32

ps_testware
Software Testing Service

Test Development

- **Nr. of user/test requirements (coverage)**
- **Nr. of business critical user/test requirements (depth)**
- **Availability and quality of test basis**
- **Use of test tools**
- **Test experience of the testers**
- **Nr. of test procedures**
- **Availability of development knowledge**



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 33

ps_testware
Software Testing Service

toc
Intro
Forecast
Manage
Finalise
Summary
Bonus
PST

Test Execution

- **Average Nr. of test data sets**
- **Nr. of test procedures**
- **Availability of automation tool**
- **Nr. of required test cycles**
- **“Resetability” of the test bed**



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 34

ps_testware
Software Testing Service

toc
Intro
Forecast
Manage
Finalise
Summary
Bonus
PST

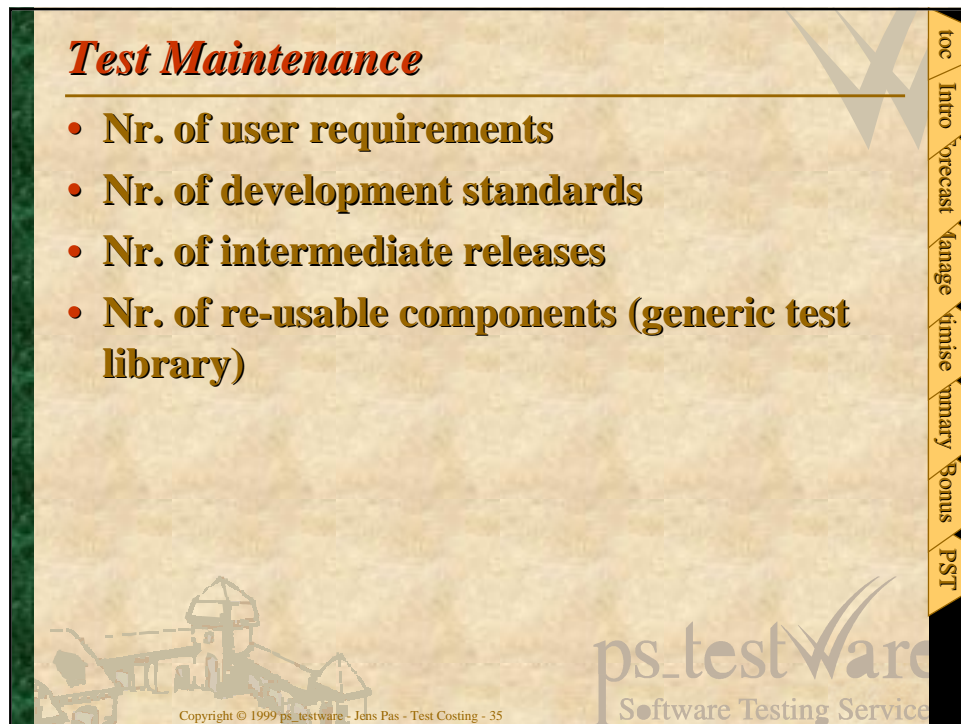
Test Maintenance

- **Nr. of user requirements**
- **Nr. of development standards**
- **Nr. of intermediate releases**
- **Nr. of re-usable components (generic test library)**

toc Intro Forecast Manage Analyse Summary Bonus PST

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 35



Test Repair

- **See test development**

toc Intro Forecast Manage Analyse Summary Bonus PST

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 36



Defect Tracking

- **Nr. of defects found**
- **Availability of a defect tracker**

toc Intro Forecast Manage Analyse Summary Bonus PST

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 37

Test Report

- **Availability and quality of test repository and access tool**

toc Intro Forecast Manage Analyse Summary Bonus PST

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 38

Overhead

- **Maturity of project management**
- **Availability and quality of project documentation**

toc Intro Forecast Manage **Finalise** Summary Bonus PST

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 39

Examples

- **Test Planning**
 - If test basis has bad quality, use workshops to define test requirements
 - **Technique switch**
- **Test Maintenance**
 - If many intermediate releases, many re-testing required => reduce intermediate release and “package” repaired software

toc Intro Forecast Manage **Finalise** Summary Bonus PST

ps_testware
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 40

Rules for ABC

- **Select Activity pools based on organisational structure**
- **Define only activity pools and cost drivers which you can easily identify and measure**
- **Use 20/80 rule: 80% of the costs come from 20% of cost drivers.**
- **Work on one cost driver at a time (theory of constraints)**
- **Be aware of correct time horizon**
- **Measure!**

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 41

toc Intro Forecast Manage Optimise Summary Bonus PST

Summary

- Testing is inevitable
- Testing has no contribution if no repair is done
- Managing Test Costs = Managing development costs
- Forecast Test Costs, using **budget and activity base ratios**
- Manage your Test Costs with **Theory of Constraints**
- Optimise your Test Costs using **Activity Based Costing**

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 42

toc Intro Forecast Manage Optimise Summary Bonus PST

Bonus: some Value Logic

| | Conventional Logic | Value Logic |
|-------------------|------------------------------------|---|
| Assumption | Development conditions are given | Development conditions can be shaped through testing |
| Assumption | Testing is not a business activity | Leverage your investments in testing and sell your testing experience |
| Assumption | ... | ... |

**Reveal hidden assumptions (Himalayas principle)
Always think in terms of value creation (throughput)**

Advised reading: W. Chan Kim, Renee Mauborgne, "Value Innovation", Harvard Business Review, Reprint 97108


Copyright © 1999 ps_testware - Jens Pas - Test Costing - 43

ps_testWare
Software Testing Service

toc Intro Pre-cast Manage Minimise Summary Bonus PST

ps_testWare
Software Testing Service

Questions



Copyright © 1999 ps_testware - Jens Pas - Test Costing - 44

toc Intro Pre-cast Manage Minimise Summary Bonus PST

ps_testWare
Software Testing Services

Tiensesteenweg 329
B-3010 Leuven
Tel.: +32 (16) 35.93.80
Fax: +32 (16) 35.93.88
e-mail: ps_testware@compuserve.com
<http://www.pstestware.com>

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 45

toc Intro Pre-cast Manage Finise Summary Bonus PST

ps_testware

- 6 • **Started in 1991**
- 5 • **1993: Tools & technical services**
 - Tool Training
 - Coaching
- 6 • **1995: Methodological Services**
 - Consultancy
- 15 • **1996: Software Testing Services**
 - Test Assignments
 - Test Plan
 - Test Report
- 25 • **1997: Software Testing Services Suite**
 - Test Assessments
 - Y2K training
- 33 • **1998: PSTI**
 - Office @NL
 - Total Outsourcing
 - Partnerships
- + 28? • **1999:**
 - ...

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 46

toc Intro Pre-cast Manage Finise Summary Bonus PST

Our Business

- **Structured Software Testing**
- **Methodology**
- **Implementation Model**

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 47

Our Services




- **Training (see ps_testware institute)**
- **Coaching**
- **Consultancy**
- **Outsourcing (now also Total Outsourcing)**

Provided by:

- Test Engineers
- Test Consultants
- Management Consultants

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 48

Our Products

- **Test Assessment**
- **Test Assignment**
- **Test Plan**
- **Test Report**
- **Test Advice**
- **Test Audit**
- **Test Pack™** 
- **Test Laboratory** 
- **Tools** 

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 49

toc Intro Forecast Manage Analyse Summary Bonus PST

References

- **Kredietbank**
- **Barco Graphics**
- **Exact Maatwerk**
- **ING Bank**
- **Bank Card Company**
- **Janssen
Pharmaceutica**
- **Tessa**
- **Europese Raad**
- **Lernout & Hauspie**
- **Origin**
- **Specs**
- **Gemeentekrediet**
- **Siemens**
- **ING 2**
- **Yokogawa**
- **Link**
- **Alcatel Bell**
- **Mobistar**

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 50

toc Intro Forecast Manage Analyse Summary Bonus PST

Credo

ps_testware's first responsibility goes to the customers who use our services. Our services must be of high quality and must be a reference for our customers. In line with our primary business, Structured Software Testing, we may not indulge in pressure, quantity or quick profit.

We are responsible to our members, the men and women who work with us. Every member must be respected as an individual and must be rewarded personally and fairly. We must support our members through a competent management, an adequate working environment and proper working conditions. Our members must have the means to provide and receive feedback, allowing them and the organisation to learn continuously. We must support our members in their family responsibilities. Our actions must be just and ethical.

Our final responsibility is to our stockholders. Our business must make a sound profit. We must innovate and continuously improve our methods and techniques. We must develop new services and implement them effectively and efficiently. We must create reserves to provide for adverse times. Our stockholders must receive a fair return on their investments.

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 51

ps_testware
Software Testing Service

toc Intro Forecast Manage Finance Summary Bonus PST

The Mission

To offer the best solution to quality problems of computer systems by using its test expert knowledge in a professional way.

Best solution: the solution that provides the highest contribution.

Test expert knowledge: the intellectual asset of ps_testware, a profound and complete knowledge regarding verification and validation (testing).

Professional: the courage to really provide what has been promised.

Copyright © 1999 ps_testware - Jens Pas - Test Costing - 52

ps_testware
Software Testing Service

toc Intro Forecast Manage Finance Summary Bonus PST

Test Costin



ROADMAPS FOR SPI

*Marina Blanco, Jose Ángel Díez, Pedro Gutiérrez,
Leire Markaida, Giuseppe Satriani.*

European Software Institute
Parque Tecnológico de Zamudio #204
E-48170 Vizcaya SPAIN

The European Software Institute (ESI) maintains one of the most important repositories of software process improvement experiences containing results of more than 200 European organisations. The present paper analyses the repository, depicts the current scenario of SPI experiences in Europe and presents a roadmap in order to help organisations plan an improvement initiative.

1. INTRODUCTION

Since 1995 the VASIE database has been continuously collecting the software process improvement (SPI) experiments funded by the European Systems and Software Initiative. Currently¹ this database contains the experience of 216 organisations in changing and improving their software processes. VASIE is permanently growing and can be accessed through ESI web site at www.esi.es/VASIE in order to download the case studies. Every month around 1400 experience reports are downloaded which indicates how VASIE is considered in the SPI world.

Sections 2 to 4 analyse in depth the contents of the VASIE database. Sections 5 to 7 explain and illustrate the *Roadmap* concept. A roadmap is a way to encapsulate the experience that exists in VASIE through the analysis of the patterns of solutions implemented by the organisations in order to achieve better technical and business results. A roadmap is intended to help organisations in the design of their own improvement initiatives based on VASIE experience.

2. WHAT IS VASIE

The European Commission (EC), being aware of the increasing technical, strategic and social importance of software, created the European Systems and Software Initiative (ESSI) with the objective of improving European industry competitiveness through the improvement of software process capability. This programme pushes software process improvement through the direct funding of improvement experiments and dissemination of the achieved results across Europe. A PIE offers

¹ September 1999

the opportunity to demonstrate the benefits of process improvement, through a controlled, limited experiment executed in a particular organisation. Dissemination actions however, are in charge of the dissemination of PIE's results.

The VASIE project (Value Added Software Information for Europe) is a dissemination action, created in 1993 with the aim of collecting, validating and disseminating the above mentioned PIE's results. In order to achieve this dissemination objective, the European Software Institute (ESI), has developed a public repository that contains the results of the ESSI Process Improvement Experiments.

From its beginnings, in 1993, until the year 2000 the ESSI programme will have funded around 400 Software Process Improvement (SPI) projects in European organisations ranging from hundred of thousands to organisations of 5 employees, generating improvement experiments in the entire spectrum of organisations. This fact makes possible that any organisation interested in improving can always find examples of similar organisations that can be used as guidelines for designing their own improvement initiatives.

VASIE is probably the most important and largest repository of SPI experiences, and is definitively the most representative of the different European Industries. It was especially designed for organisations thinking of designing an improvement project, in order to allow them to learn how others had done so, the problems they had encountered, and the solutions they had provided.

Thanks to the high diversity of organisations that have been performing experiments under ESSI, any organisation interested in software process improvement can find experiences of similar organisations, solving similar problems and so analyse them before designing their own improvement programme. All the PIEs are stored and classified in a database. The set of searches designed to access the database allows users either to locate experiments performed in specific contexts (country, industrial sector, organisation size, software department size, etc.), or to locate experiments that solve particular problems (high number of errors after delivery, project time deviation, etc.).

Users can read information reported directly by the organisations involved in the SPI experiments. This information can be read in a HTML browser or downloaded in PDF format. All the reports clearly indicate which is the organisation contact information in order to facilitate the access to first hand information provided by the main actors of the experience.

VASIE is currently being used by a large number of organisations that browse around 600 reports in HTML and download around 800 reports in PDF format every month i.e. 1400 reports in total. These encouraging figures are an indicator of the acceptance of VASIE as an active instrument for the design of SPI projects for

supporting consultants activity and for extracting case-studies to be discussed in a classroom.

3. WHAT IS A PIE

A PIE aims at demonstrating the benefits of improving the software development process through a controlled limited experiment. A PIE (see Figure 1) is usually undertaken in conjunction with a real software project (the baseline project), which forms part of the normal business of the organisation. A PIE allows an organisation to try out new procedures, new technologies and organisational changes before taking the decision as to whether or not they should be replicated throughout the software-developing unit.

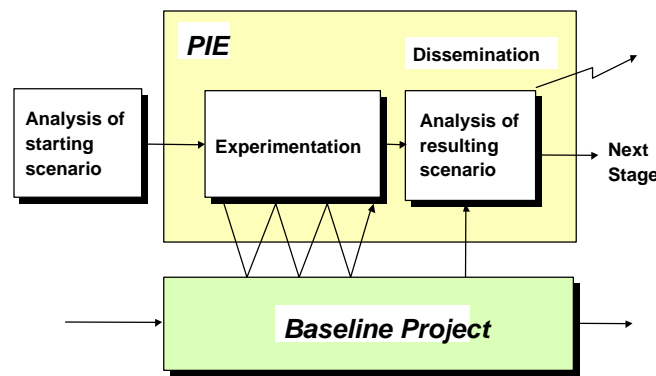


Figure 1 A Process Improvement Experiment (PIE)

There is a very specific information flow between a PIE and VASIE (see Figure 2), in order to capture all relevant information.

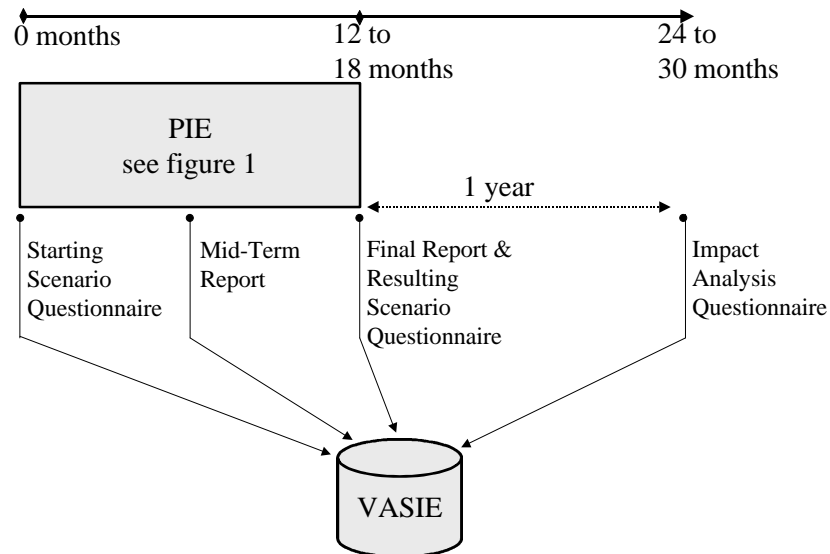


Figure 2 Information flow between the PIE and VASIE.

Starting Scenario Questionnaire. The PIE send this questionnaire at the very beginning, informing about the organisations that participate in the PIE, their context, the objectives they pursue, and the problems they want to tackle.

Mid-term Report. Reflects the activities performed by the PIE in the middle of the experiment.

Final Report. The complete story of the experiment is submitted at the end of the PIE and summarises problems, solutions, implementation steps, difficulties found, benefits achieved, lessons learned, etc.

Resulting Scenario Questionnaire. Submitted together with the Final Report, reflects the processes changed, facilitator and inhibitors encountered, etc.

Impact Analysis Questionnaire. Approximately one year after the end of the PIE, another questionnaire is submitted that reflects the long-term benefits achieved, and follow up activities started.

Every PIE goes through a Review Process that decides whether the Final Report contains relevant information, reproducible solutions and valuable lessons and benefits. Those experiments that fulfil the criteria established for the Review Process are inserted in the repository for public dissemination, in addition the final report of each PIE has to follow a clear and logical structure, and provide an adequate clarity and level of conciseness.

The PIEs are classified according to the organisation context and to the own experiment context. In relation to the organisation context, the most relevant parameters are: the country, the organisation size, software department size, and the industrial sector. The experiment context is described using parameters such as the business goals pursued by the organisation, the most common problems that the organisation expected to solve performing the improvement action and the software processes affected by the experiment. Each final report describes these parameters and the executed tasks during the experiment, the procedures, templates, tools and methodologies used. And finally the obtained results and the lessons learnt are explained to help other organisations with similar experiments.

4. VASIE CONTENT OVERVIEW

In September 1999, VASIE contains 184 PIEs, and due to the fact that more than one organisation can participate in a PIE the repository includes about 216 organisations experiences.

The complete set of questionnaires obtained from the PIEs let VASIE characterise them in depth. The analysis of the information contained in VASIE allows to figure out the SPI picture in Europe with high precision

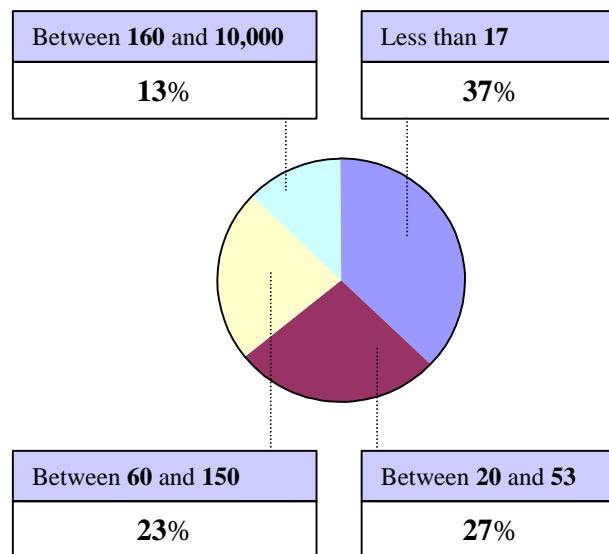


Figure 3 Software development unit size distribution (Number of employees)

The PIEs have been fundamentally executed in small software development units. In fact, software development units of less than 17 employees have carried out 37,35% of the experiences. The figure below shows how the groups of units under 17 employees and the group from 17 to 52 are the two most important with respect to the number of PIEs executed.

Not all the industrial sectors have been equally covered by ESSI. See Figure 4 for detailed information of the most representative industrial sectors.

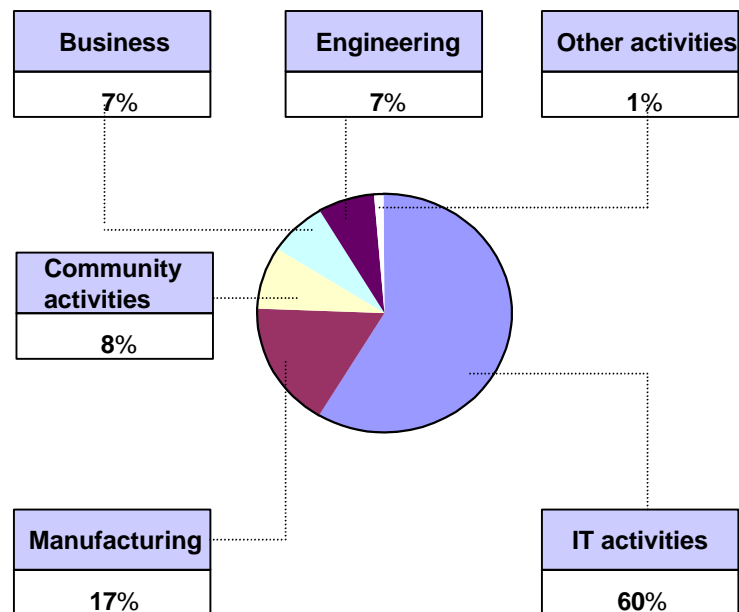


Figure 4 Percentage of organisations per activity in VASIE.

Figure 5 illustrates how the different European countries are represented in VASIE with respect to the number of suitable PIEs executed.

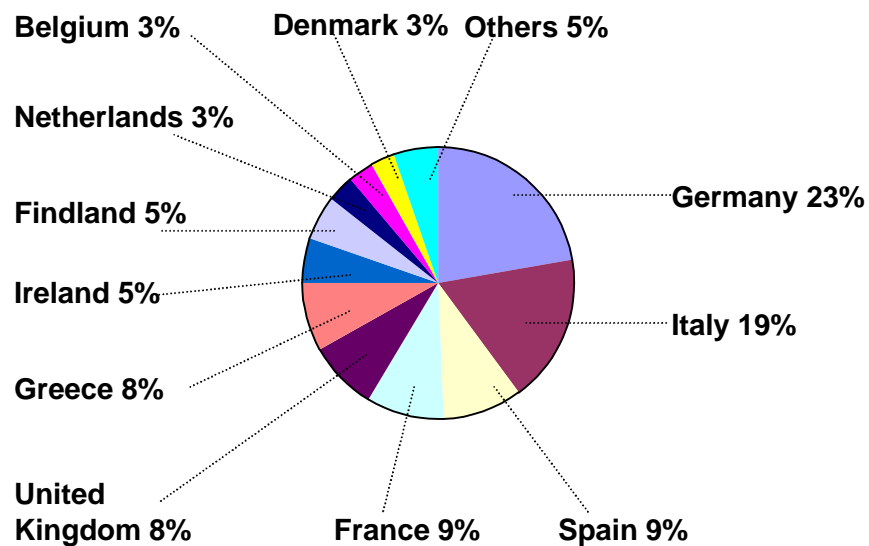


Figure 5 Countries represented in VASIE.

In order to categorise the different PIEs in VASIE according to the capability level of the organisations where they were executed, Figure 6 shows the capability of the organisations before the PIE start.

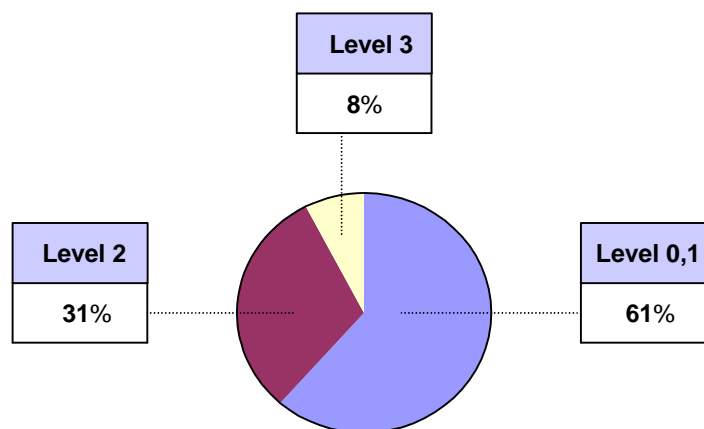


Figure 6 Distribution of organisations per capability level in VASIE

- Levels 0 and 1 indicate that there are evidences of a lack or ineffective use of project management.
- Level 2 means that there are evidences of the existence of a project management function, at least at project level.

- Level 3 means that there are evidences of the existence of organisational procedures and methods that are used by the software projects according to a defined software development life cycle at organisational level.

A different view of the capability of the organisations involved in the PIEs could be observed in Figure 7 taking into consideration their state with respect to the ISO 9001 certification before the PIE start.

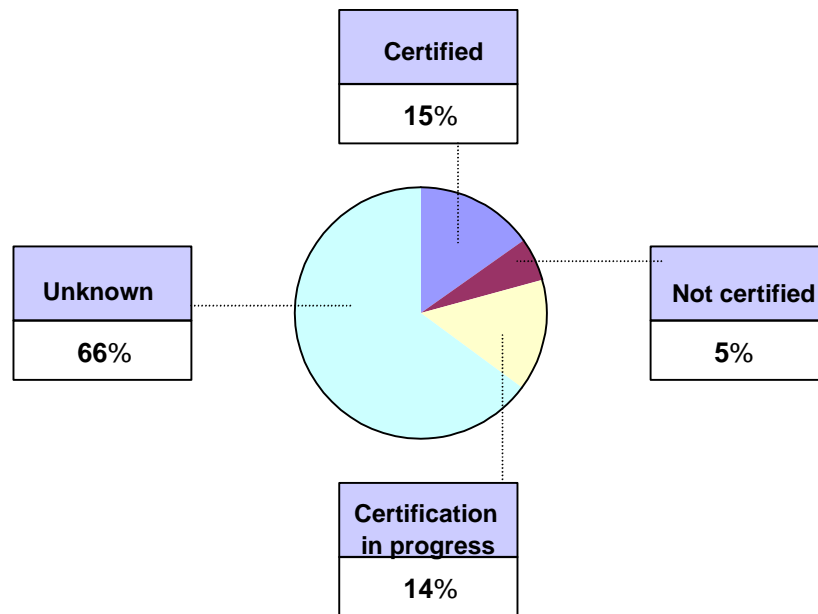


Figure 7 Status with respect to ISO 9001 in VASIE

Table 1 indicates which are the three business goals most representative in VASIE as stated by the organisations involved in the PIEs.

| Place | Business goals pursued |
|-----------------|---------------------------|
| 1 st | Improve delivered quality |
| 2 nd | Cost reduction |
| 3 rd | Increase productivity |

Table 1 Business goals most frequently pursued by organisations in VASIE

Table 2 summarises the most frequently tackled problems in the PIEs, ordered by the importance assigned to each problem,.

| Place | Problems tackled |
|-----------------|----------------------------|
| 1 st | Reusability |
| 2 nd | Project cost deviation |
| 3 rd | Lack of process definition |

Table 2 Problems most frequently tackled by organisation in VASIE

And finally, Figure 8 shows the improvement areas the PIEs preferred to improve:

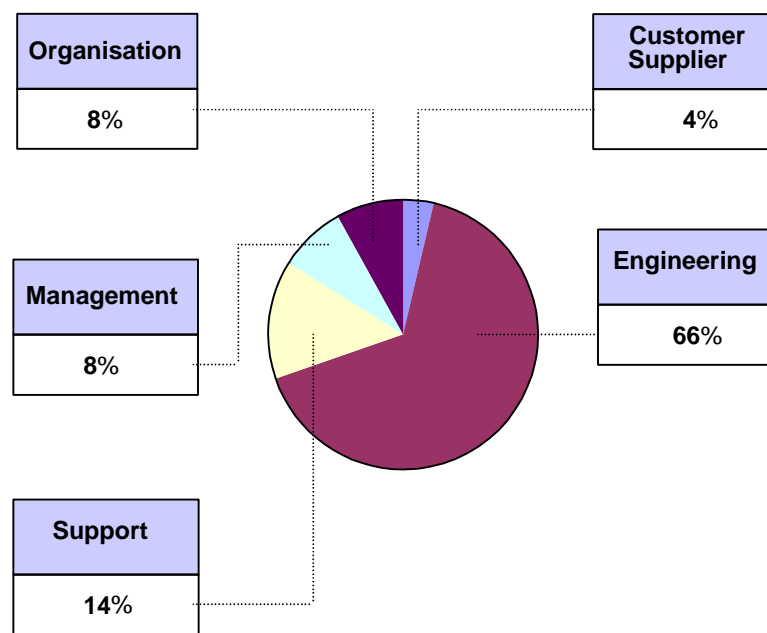


Figure 8 Distribution of improvement areas preferred by organisations in VASIE.

Each area corresponds to a process category as specified by ISO 15504 (SPICE); The following definitions have been extracted from it.

“The Customer-Supplier improvement area consists of processes that directly impact the customer, support development and transition of the software to the customer, and provide for the correct operation and use of the software product and/or service.

The Engineering improvement area consists of processes that directly specify, implement or maintain the software product, its relation to the system and its customer documentation. In circumstances where the system is composed totally of software, the Engineering processes deal only with the construction and maintenance of such software.

The Organisation improvement area consists of processes that establish the business goals of the organisation and develop process, product, and resource assets which, when used by the projects in the organisation, help the organisation achieve its business goals. Although organisational operations in general have a much broader scope than that of software process, software processes are implemented in a business context and, to be effective, require an appropriate organisational environment.

The Support improvement area consists of processes that may be employed by any of the other processes (including other supporting processes) at various points in the software life cycle.

The Management improvement area consists of processes that contain practices of a generic nature that may be used by anyone who manages any type of project or process within a software life cycle.”

5. ROADMAPS

Analysis of the data contained in the VASIE repository shows the existence of patterns of solutions which are commonly applied by many organisations. Roadmaps are, within the context of this paper, a mechanism to represent those patterns of solutions (the roads) which organisations in VASIE have implemented in order to reach the business and technical goals² (the destinies). A solution is represented in terms of the set of ISO 15504 processes improved in order to reach a goal.

Solutions are classified into different categories according to their level of reliability. Higher reliability levels are assigned to solutions which have been tested in real production projects, while lower levels are assigned to solutions tested in laboratory projects or off-line environments.

Roadmaps provide a picture of the different solutions available to reach a specific goal. They allow the evaluation of alternative solutions, their level of reliability, the type of organisations that have used them, and many other factors that facilitate the selection of the most appropriate route for the particular context of an organisation.

The following are some tips to better understand a roadmap:

- If a road crosses different processes, it can not be deduced that the organisations changed the processes in the order the road crosses those processes. That is if a road crosses first through *software design* and then through *software requirements*, it can no be deduced that an organisation improved first the design

² These business and technical goals are collected at the very beginning of the PIE, though the Starting Scenario Questionnaire (already described in section 3).

and afterwards the requirements process. Both processes were improved, but nothing can be said about the order.

- The road width is an indicator of the number of organisations that used that solution; i.e. the road traffic.
- The road length (at the moment) is not an indicator of the effort to implement the solution.
- The processes represented are processes the organisations improved in order to reach a goal. However not all the processes changed are represented in the roadmap. Processes with low traffic have been omitted for the sake of simplicity.
- A roadmap (at the moment) does not allow the judgement of the success of any of the solutions adopted.

6. A ROADMAP

The roadmap presented in this paper (see Figure 9) is an example of the different roadmaps that are being derived from VASIE. This one analyses the experiences of 18 organisations. These organisations shared the common goal of *increasing the delivered quality of their software products*, and the roadmap highlights the solutions they used. Two solution alternatives have been followed. The first alternative followed by group A, that consists of 9 organisations, improves the following processes:

- Software design and
- System and software requirements.

The second alternative followed by group B, that consists of 9 organisations, improves the following processes:

- Integration & testing of system and software
- Project management
- Verification.

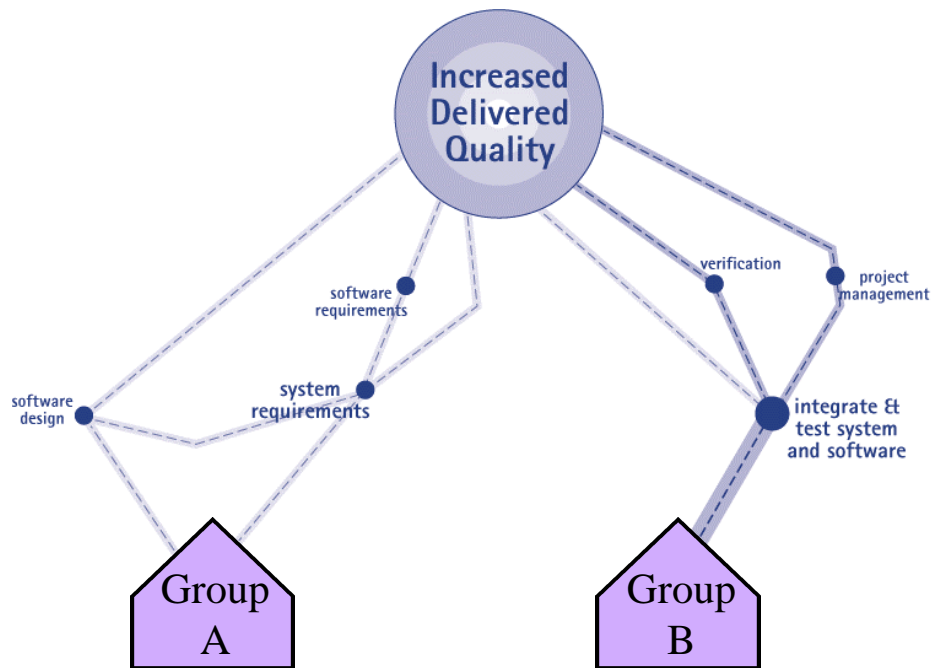


Figure 9 Two different alternatives to increase delivered quality

According to the roadmap above, group A focuses on lifecycle processes, specially on the initial processes of the cycle. However, group B put a tremendous³ attention on one of the final processes of the cycle, i.e. the integration and test of both software and system. What is more, group B goes beyond the lifecycle and improves process such as *project management* and *verification*.

For a better comprehension of the reasons why these two sets of organisations implemented different solutions, it is interesting to go into the details of the context of both groups. The context is explained in Table 3 through the following characteristics:

- *Maturity level*; as described in section 4
- *Primary involvement in software industry* i.e. the role each organisation performs in software industry.
- *Baseline Project characteristics*; as mentioned in section 3 a PIE is implemented in conjunction with a real software project. This project is said to be a *laboratory* project when it is an off-line environment that resembles the actual software

³ According to the high traffic the roadmap represents for that process

development environment. A project is said to be a production project when its primary objective is to deliver a software product to a customer.

- *Industrial Sector*
- *ISO 9001 certification*

| | GROUP A | GROUP B |
|---|---|--|
| Maturity Level | <p>Level 2 13% Level 1 87%</p> | <p>Level 3 11% Level 1 22% Level 2 67%</p> |
| Primary Involvement in Software Industry | <p>Software User (primarily developed by a 3rd-party) 11% Software User (primarily developed in-house) 89%</p> | <p>Software Vendor (producing custom software systems) 15% Software User (primarily developed by a 3rd-party) 8% Software Vendor (producing off-the-shelf systems) 23% Software User (primarily developed in-house) 54%</p> |
| Baseline Projects Characteristics | <p>Production 13% Laboratory 87%</p> | <p>Laboratory 33% Production 67%</p> |
| Industrial Sectors | <p>Textile and Textile products 13% Agriculture and Forestry 13% Software Consultancy and Supply 13% Machinery, Electrical and Optical Instruments 24% Telecommunication Products 13% Electronic Components 24%</p> | <p>Electrical Engineering 11% Mechanical Engineering 11% Telecommunication Products 11% Finance and Insurance 11% Health 11% Machinery, Electrical and Optical Instruments 23% Software Consultancy and Supply 22%</p> |
| ISO 9001 | <p>Certified 13% In the Certification Process 13% Unknown 74%</p> | <p>Not Certified 11% Certified 56% Unknown 33%</p> |

Table 3 Roadmap groups characteristics.

Table 3 data suggest that organisations at level 2 and organisations that are ISO 9001 certified prefer the solution followed by group B while level 1, and not certified organisations prefer the other solution.

Another important characteristic highlighted by the table about the reliability level of each solution, is that group B solution is tested in more realistic baseline projects i.e. production projects. And it could be expected that the results of that solution are more realistic too.

7. CONCLUSIONS

The Roadmap presented in this paper summarises the patterns of solutions implemented by the organisations in order to reach a goal. However, it could be interesting to understand whether the solution helped the organisations reach the goal, or to what extent. It could be also interesting to know which is the average effort associated with the implementation of each solution. These and other information are crucial to decide which solution to implement and without them roadmaps are not so useful. ESI is currently deriving improved roadmaps, which provide this useful information, using the *Impact Analysis Questionnaire* (see section 3). This questionnaire, filled in by the organisation a year after the PIE ended provides visibility on the long term goals affected by the PIE and about the effort invested, as long as visibility on other important data.

8. BIBLIOGRAPHY

Marina Blanco, Pedro Gutiérrez, Leire Markaida, Giuseppe Satriani (1999) *Repository of Expertise Guide*. European Software Institute. ESI-1999-TR-017

ESI (1995) *The VASIE Repository* at www.esi.es/VASIE

Marina Blanco, Jose Ángel Díez, Pedro Gutiérrez, Leire Markaida, Giuseppe Satriani (1999). *Software Process Improvement Driven by Business Goals: The role of experiences*. Sixth European Conference On Software Quality: Software Quality-The Way To Excellence (Vienna, Austria, April 12-16, 1999)

ISO (1998) *ISO/IEC 15504: Software Process Assessment: Parts 1 - 9*, Technical Report - Type 2 (approved for publication)

Herbsleb, James et al (1994) *Benefits Of CMM-Based Software Process Improvement: Initial Results*. Software Engineering Institute. CMU/SEI-94-TR-13

ESI

Roadmaps for SPI

Pedro Gutierrez
3rd International Software Quality Week Europe

European Software Institute
Parque Tecnologico de Zamudio #204
Bizkaia, Spain

ESI

European Software Institute

ESI's Style

Non-profit association, created in 1993

Supported by the European Commission , the Basque
Government and through company membership

ESI's Mission

To support our members and European industry to improve
competitiveness by promoting and disseminating best
practice in software

ESI's Focus

Software Process Improvement to optimise business results

Agenda

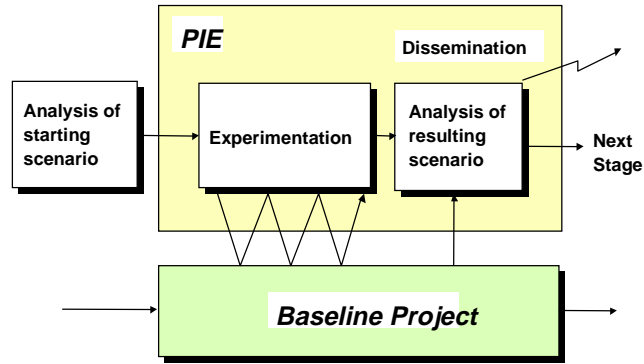
- **VASIE**
Created in 1995,
Collects the software process improvement case studies
funded by the ESSI during the last 7 years.
- **Roadmaps for SPI**
Encapsulate the knowledge that exist in VASIE through the
analysis of patterns of solutions.

Some details

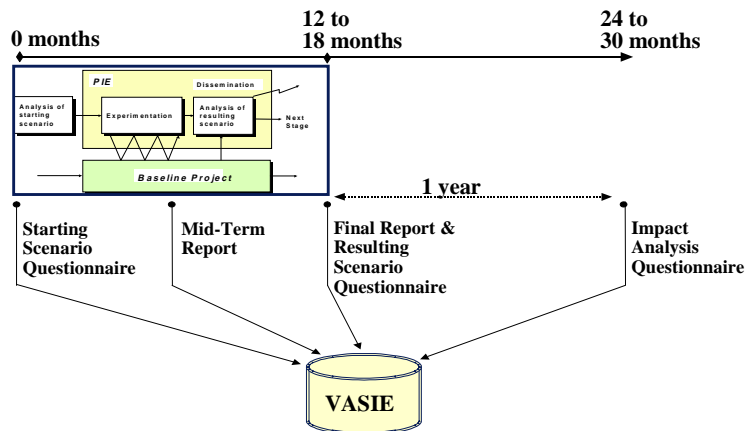
- 216 organisations represented in VASIE grouped
in 184 case studies
- 1400 case studies downloaded monthly
- Public database

www.esi.es/VASIE

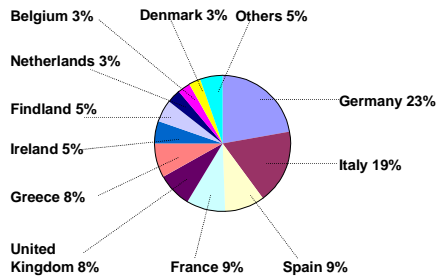
PIE: Process Improvement Experiment



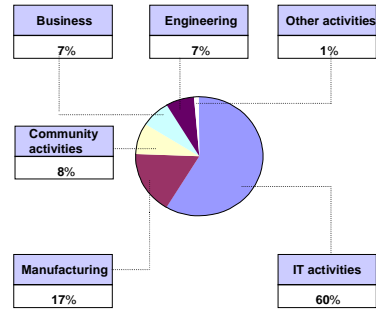
Information flow between VASIE and the PIE.



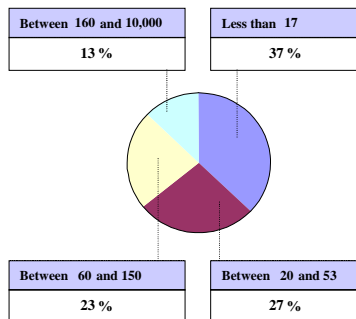
Country



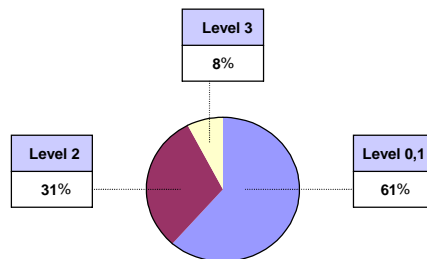
Industrial Sector



SW Department Size



Maturity Level



Business Goals

1st Improve Delivered Quality

2nd Cost Reduction

3rd Increase Productivity

Problems Tackled

1st Reusability

2nd Project Cost Deviation

3rd Lack of Process Definition

VASIE Main Objective

Facilitate the access to case studies

- Using the WWW
- Powerful Search Mechanism
 - Business Goals
 - Organisation Context
 - Problems Tackled
 - Business Goals

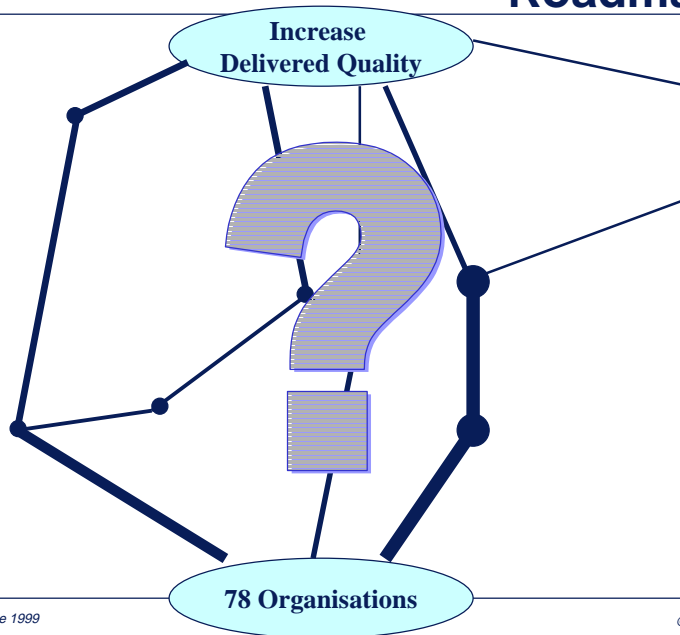


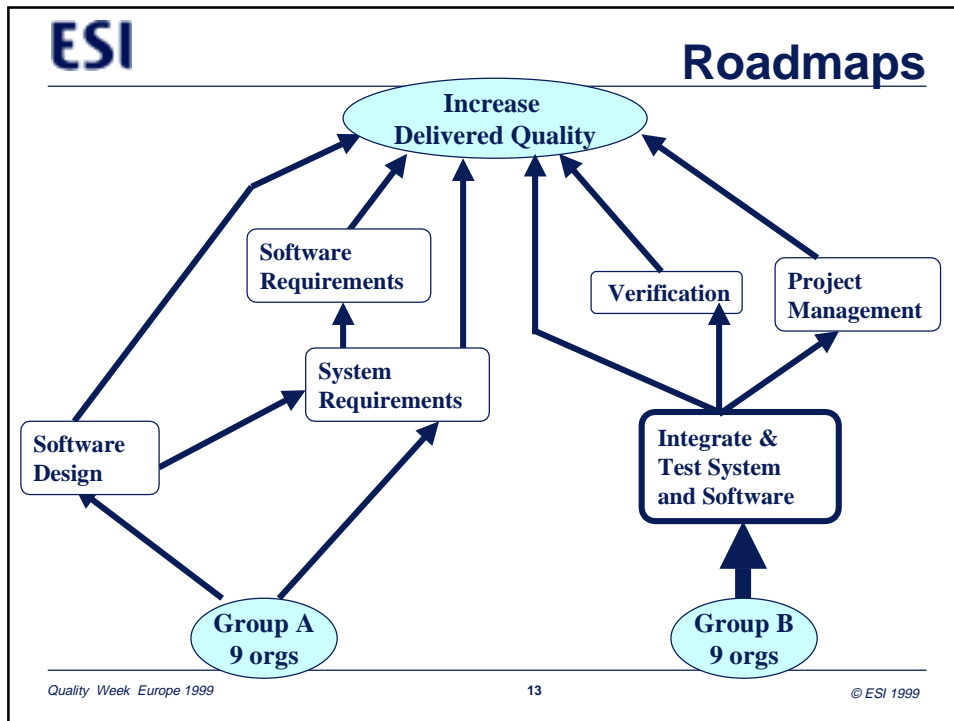
VASIE 2nd Objective

Encapsulate VASIE knowledge in a way that supports **decision making**

Roadmaps

- Capture explicit patterns of solutions into roads
- Easy to interpret





ESI **Roadmaps**

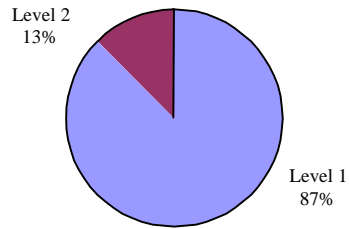
Context Analysis

- Industrial Sectors
- ★ Maturity levels
- ★ ISO 9001 certification
- Role in software industry
- Size
- ★ How they implemented the improvement

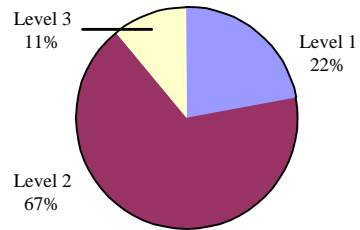
Quality Week Europe 1999 14 © ESI 1999

Maturity Level

Group A

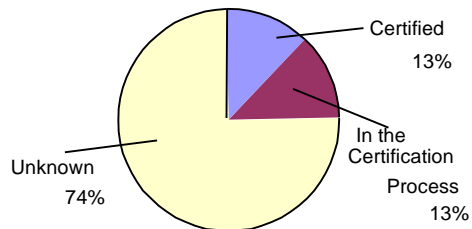


Group B

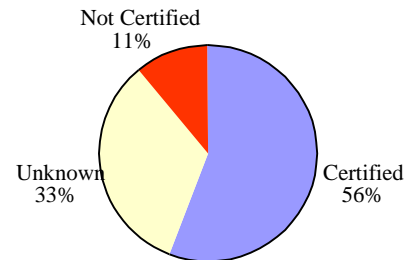


ISO 9001 Certification

Group A

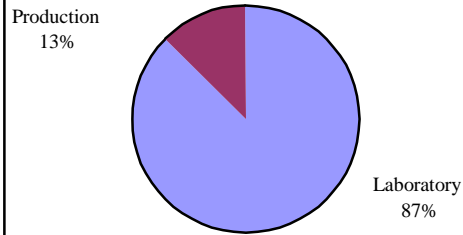


Group B

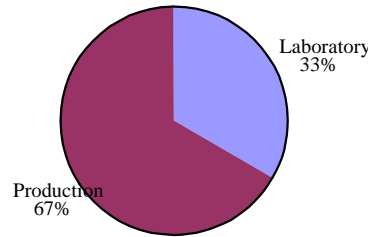


Baseline Projects Selection

Group A



Group B



Summarising

Group A

- **Maturity level 1**
- **Not certified**
- **PIEs in laboratory baseline projects**

Group B

- **Maturity level 2**
- **Usually certified**
- **PIEs in production projects.**

Limitations

- It is not clear whether the business goal is achieved or not.
- There are no indications about the cost of the implementation of a solution.

Current and Next Steps

Analysis of the Impact Analysis Questionnaire

- Clarify whether the goal is reached and to what extent
- Average effort to implement one solution

Achieving Business Excellence in SPI: Applying the EFQM/SPICE Integrated Model in Industry

Luigi Buglione & Elixabete Ostolaza

EUROPEAN SOFTWARE INSTITUTE

Parque Tecnológico de Zamudio #204

E-48170 Zamudio, Bizkaia, Spain

E-mail: {luigi, elixabete}@esi.es

Tel: (34) 94 420 9519 - Fax: (34) 94 420 9420

Index – 1. Introduction - 2. The EFQM/SPICE Integrated Model 1998 version – 3. The Assessment Method – 4. Trials 1998 – 5. Lessons Learned from Trials 1998 – 6. Trials 1999 – 7. Conclusions and prospects – References

Abstract – *This work analyses the increasing importance for a Software Intensive Organisation (SIO) to implement and take advantage of a TQM model tailored for the software field in order to achieve business excellence through continuous improvement.*

ESI has developed the EFQM/SPICE Integrated Model, that combines the strengths of two well-known and accepted models: EFQM - for the business side, taking advantage of its capability to link results with specific business goals through a causal chain - and SPICE (ISO/IEC 15504) - for the software improvement side, taking advantage of its effective process-based approach.

Additionally, an Assessment Method was developed to provide SIOs with a well-defined and structured approach to evaluating the degree of excellence of their process performance and to its deployment throughout the organisation, describing in detail the phases of the assessment and the activities and tasks to be performed, based on an ETVX (Entry-Task-Validation-eXit) process definition notation approach.

Last year this three-level tier method was tested by ESI. Trial sessions were carried out on some Spanish SIOs in order to verify the adequacy and consistency of the 1998 version of the model, as a necessary step to bridging the gap between theory and practice.

The experience gained from these trials is presented here, as well as the consequent suggestions and useful starting points for the improvements of the upcoming model evolution.

The new 1999 Trials structure is also presented.

Keywords – EFQM Model, SPICE, TQM, Information Tecnology, Software Intensive Organisations, Assessment Method.

Conference Topics – Process Assessment/Improvement; Mature Software Processes; Real-World Experience

1. Introduction

Over the last few years, process improvement has demonstrated that a company's business results are based on how the organisation manages its processes. Software Process Improvement (SPI) therefore has a much wider relevance than just its software engineering technical aspects. An analysis of the relationship between SPI and Total Quality Management (TQM¹) is now viewed as a prerequisite for a successful improvement programme in a Software Intensive Organisation (SIO)². Figure 1 shows this relationship referring in particular to CMM that, as stressed by Paulk [9], covers just the process side of TQM³ specifically for software engineering, deliberately not addressing other relevant aspects, like people issues⁴. In fact, traditional software process improvement models are not TQM oriented. They manage software processes as isolated from the rest of organisation's key processes. These models focus on measuring the process in order to control it and guarantee its continuous improvement. However, they lose the global picture of the company and often measures are not tied to business goals. On the other hand, TQM models like EFQM and Malcolm Baldrige do not provide any help in

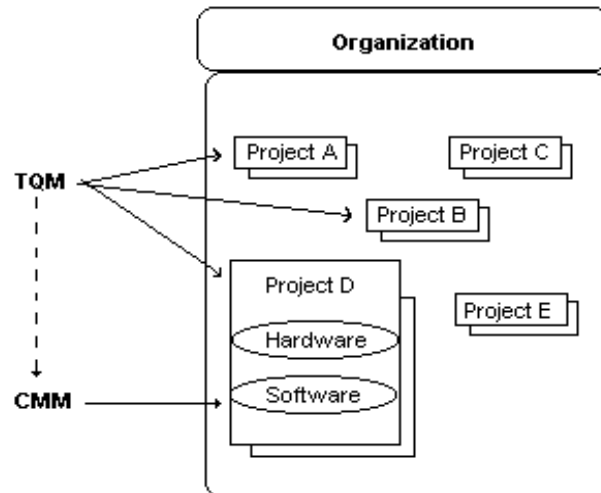


Figure 1 - Relationship between a SPI framework (CMM) and TQM [11]

the difficult task of defining the key processes for a software organisation and how to improve them to achieve concrete business goals. In order to help overcome these problems, the European Software Institute (ESI) has developed the EFQM/SPICE Integrated Model [6]. The EFQM/SPICE Integrated Model is a process-based model for continuous improvement that integrates SPI into the wider context of TQM. For this reason it could be considered as the business excellence road for software intensive organisations. It is the result of merging two well-known and accepted models: SPICE and EFQM:

- The European Foundation for Quality Management (EFQM) and its Excellence Model [5] take into account the Total Quality concepts;

¹ Kanji & Asher [8] give an interesting distinction between the concepts of quality and TQM, affirming that "Quality is to satisfy customers' requirements continually; TQM is to achieve quality at low cost by involving everyone's daily commitment".

² SIOs are organisations whose main objective is software development and selling or software departments of organisations that develop software as integrating part of its final products or organisations that develop software for internal use to achieve better business results or whose software department can be qualified as an independent organisational unit.

³ Covey [3] affirms that "Total Quality is an expression of the need for continuous improvement in four areas: 1) personal and professional development; 2) interpersonal relations; 3) managerial effectiveness; and 4) organisational productivity". Silver [12] lists several CMM flaws, from the reduced importance given to processes' cultural dimension to the lack of quantitative process-performance metrics.

⁴ Böttcher [1] identifies key differences between CMM and TQM after an analytic summary of their characteristics. It must be stressed that SPI shares many of the common goals and success factors which characterise not only TQM, but also the Business Process Reengineering and Learning Organisation management approaches <<http://www.objectif.fr/~spire/pages/section4.html>>.

- SPICE - Software Process Improvement and Capability dEtermination – (ISO/IEC 15504) [7] is a continuous model for process improvement containing good practices for software engineering.

The EFQM/SPICE Integrated Model provides a wider and holistic approach by emphasising the business results of process improvement. In this way, quality means meeting customer expectations, not just conformance to a model. Quality systems and business management should not be separate activities but should operate concurrently as part of the overall business system so that they are not inconsistent. Quality is not something that should be added on to a business: the whole business should be managed in a quality way.

The EFQM/SPICE Integrated Model has also inherited from the EFQM Model its important focus on stakeholders: customers – internal and external, current and potential; employees; shareholders; society and subcontractors. These stakeholders are not sufficiently considered by other models or are only considered as part of a contractual relationship. In the latter case the organisation tries to be conformant to a set of requirements rather than caring about customer satisfaction. The new model is based on concepts like partnership and establishing win-win relationships with the stakeholders. It recognises the organisation’s people as one of its most valuable assets (this is especially true for software intensive organisations). It focuses on customer satisfaction and not just in conforming to customer requirements. This is considered essential for achieving long-term company success.

This paper is subdivided into five main sections. Section 2 presents the 1998 version of the EFQM/SPICE Integrated Model. Section 3 illustrates the Assessment Method, the tool used to determine SIOs strengths and areas of improvements in their work to continuously improve processes towards business excellence. Sections 4 and 5 introduce most relevant aspects and conclusions from the 1998 trials and lessons learned, while Section 6 illustrates the 1999 trials updated design derived from ESI past experience.

2. The EFQM/SPICE Integrated Model 1998 version

The EFQM/SPICE Integrated Model v1 [6], represented in Figure 2 is the result of combining the strengths of two well-known and accepted models: SPICE and EFQM⁵. The Integrated Model maintains the wider external structure of EFQM but is internally configured like SPICE, based on processes, base practices and work-products. There is a process category for each enabler criteria and in each of these categories there are as many processes as there are sub-criteria for the relevant enabler criterion. SPICE processes play an important role in the model as candidates for key processes. A mapping between the candidate SPICE processes and the business goals will determine the key software processes for the organisation. Like SPICE, the integrated model has two dimensions:

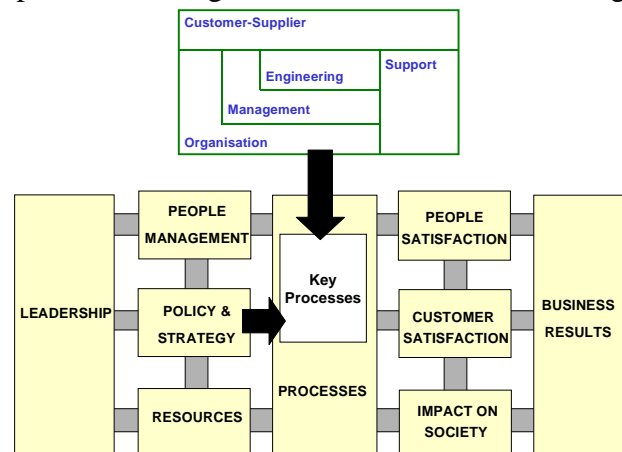


Figure 2 - Structure of the Integrated Model v1

⁵ The EFQM/SPICE Integrated Model v1 refers to the EFQM Model for Business Excellence [4].

- **Processes and Results dimension:** a descriptive representation of both the processes that a SIO aiming at TQM should implement and the quantitative data it should gather to check that it is achieving the right results. On the **Process** side, there are three types of processes (Figure 3): enabler processes, key processes and the measurement process⁶, sharing a common process structure⁷.

On the **Results** side, the Integrated Model contains a set of results, grouped by type and subtypes, to which the measurement process is applied⁸.

- **Capability dimension:** a mechanism to measure the maturity of the processes, the excellence of the results, and the scope of these maturity and excellence levels. This second dimension, the capability dimension, has two aspects: one for processes and another for results.

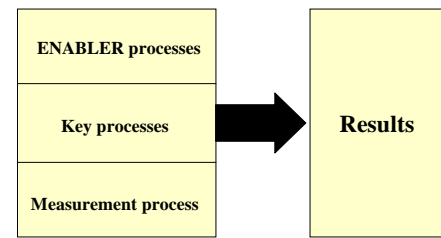


Figure 3 – Structure of the Processes and Results Dimension

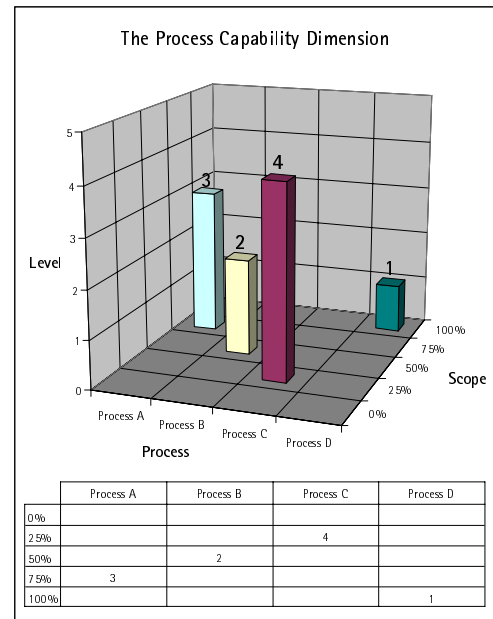


Figure 4 -The Process Capability Dimension

⁶ In detail:

- **Enabler processes:** The EFQM/SPICE Integrated Model consists of five process categories, one for each EFQM enabler criteria. Each process category has as many processes as sub-criteria in the corresponding enabler criteria of the EFQM Model;
- **Key processes:** The model provides a set of candidate processes directly extracted from SPICE; processes that characterise software organisations. Processes that are already covered by other enabler processes (corresponding to other EFQM criteria) are not under the key process candidates type;
- **Measurement process:** the measurement process identifies, collects and analyses data relating to the organisation's results. Its purpose is to demonstrate objectively the performance of the organisation in achieving its business goals and satisfying the stakeholders.

⁷ The five common elements are:

- Process name
- Process purpose
- Process outcomes
- Base practices
- Work products (inputs and outputs).

⁸ In particular:

- Each type of result corresponds to an EFQM result criterion and has two subtypes: the *perception* subtype (which contains a list of attributes to measure the perception of the results by the customer, people or society) and the *performance* subtype (with a list of attributes that measure the performance of the organisation in satisfying the needs of customers, people, society and shareholders);
- Each subtype of result maps with an EFQM result sub-criterion.

The ‘**process capability dimension**’ is defined on the basis of the combination of two factors, Level and Scope⁹ (Figure 4), while the ‘**results excellence dimension**’ is also defined on the basis of the combination of two factors, Excellence and Scope¹⁰ (Figure 5).

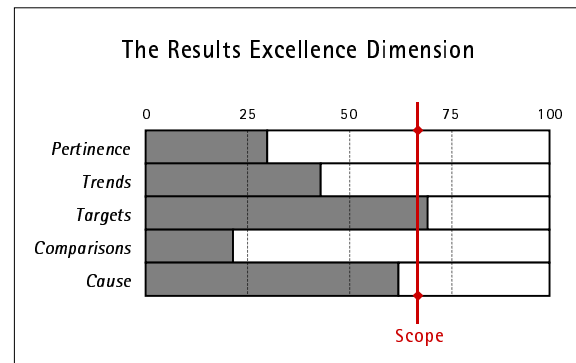


Figure 5 - The Results Excellence Dimension

3. The Assessment Method

The EFQM/SPICE Integrated Model is a static reference point, a structured group of good practices for organisation and process management, that allows top management to understand what TQM means for their software company. Applying this model to an organisation implies combining it with a set of dynamic tools and techniques that integrate TQM and SPI into business management.

The EFQM/SPICE Integrated Model could be used solely for assessment purposes or for business planning and improvement. The latter incorporates assessment, and particularly self-assessment, as a strategic step in business management based on TQM concepts. To support the application of the model, ESI has developed an Assessment Methodology [13] defining the phases of the assessment and the activities and tasks to be performed within each phase. The Method contains four first-level phases, that represent the logical workflow of an assessment, where the output of the n-th phase represents the input for the (n+1)-th one:

- 1) **Assessment Preparation**¹¹: these activities include preparing everything previous to assessing the organisational unit, from agreeing the terms of the assessment with the sponsor and collecting all the necessary information about the assessment context, to preparing the detailed plan of the work to be carried out and obtaining a qualified assessment team;
- 2) **Assessment Execution**: these activities include collecting an accurate and comprehensive set of data about the organisational unit, so that it is possible to cover the assessment purpose. As a minimum, a list of strengths and areas for improvement of the organisational unit will be produced;
- 3) **Assessment Reporting**: these activities include preparing a report of the assessment results and to present it to the appropriate people of the organisational unit;

⁹ In detail:

- LEVEL – measures the degree of excellence of the process performance and implementation approach. Like SPICE, there is a six point ordinal scale that enables capability to be assessed from the bottom of the scale, Incomplete, through to the top of the scale, Optimising;
- SCOPE – measures the degree of deployment of the process throughout the organisation. Like EFQM, there is a percentage scale that enables the scope of the application to be assessed from ‘not applied in relevant areas’ to ‘applied in all relevant areas and activities’.

¹⁰ In detail:

- EXCELLENCE - measures the excellence of the results based upon a set of five attributes: *Pertinence, Trends, Targets, Comparisons and Cause*. Like EFQM, there is a one hundred point cardinal scale that enables excellence to be assessed from the bottom of the scale (0), *No Results*, through to the top of the scale (100), *Best in Class*.
- SCOPE – measures the degree of coverage of the organisation’s relevant missions, business goals, areas and activities. There is a percentage scale that enables scope of the results to be assessed from ‘not applied in relevant areas’ to ‘applied in all relevant areas’.

¹¹ This phase and its steps are based on the ideas described in [2].

4) Assessment Follow-up: these activities include drawing conclusions from the assessment experience at ESI to reuse good practices in further experience.

Each top-level phase is broken up into a series of a total of 10 second-level activities, that can be decomposed into a series of a total of 17 third-level tasks. Each bottom-level task is defined through the objective, entry and exit criteria, activities, inputs and outputs; responsibilities and general considerations¹². The Assessment Method is supported by a full set of assessment materials including presentations, document templates, checklist and data collection forms. Figure 6 presents the 17 bottom-level tasks, describing also the workflow of the EFQM/SPICE Assessment.

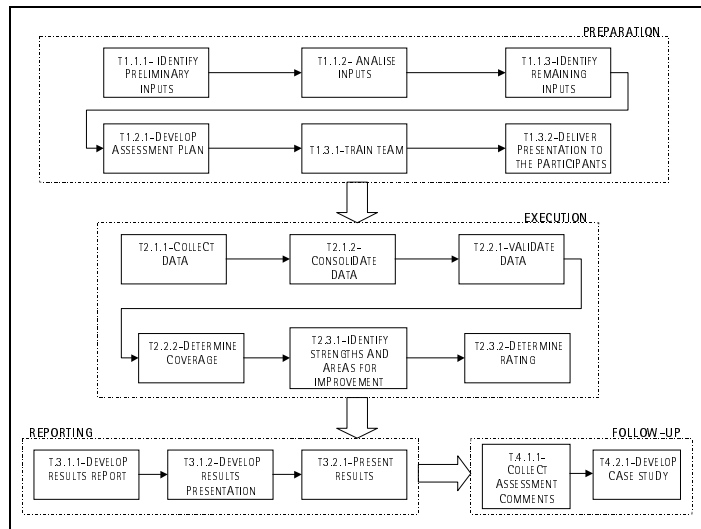


Figure 6 - Bottom-level activities of the ESI EFQM/SPICE Assessment Method

4. Trials 1998

In the second semester of 1998, ESI carried out two trials of the EFQM/SPICE Integrated Model in software departments of two large organisations. ESI trial objectives were to validate the efficiency of the EFQM/SPICE Integrated Model in a real context and to evaluate the EFQM/SPICE assessment method, while software departments objectives were to obtain a picture of their actual situation and to use the assessment results as basis for developing the SIO plan to achieve the organisation business objectives.

Both organisations are successful and experienced an important income increase in the last few years. Organisation A is already involved in an EFQM improvement initiative and its EFQM score is above 400 points (according to an official assessment carried out by the Basque Quality Foundation, Euskalit, in 1997). The organisation is composed by several branches, each of them with its own software department. The software department of one of these branches was assessed. Organisation B was not involved in any formal improvement initiative. Its central software department was assessed. The assessments were carried out following the 17 tasks of the EFQM/SPICE Assessment Method step by step. A short list of the most relevant aspects of the assessments performed summarised by phases is here described:

- **PREPARATION PHASE.** The main objective of the preparation phase is to assess the context of the organisational unit in order to know the key processes of the organisation, including the software key processes, to select later on 1) the set of ENABLER processes, SPICE processes – from the SPICE processes candidate to key processes list – and results that will be assessed, 2) the most appropriate projects to compare with and, 3) the appropriate people to interview. These basic elements, the three of them, define the assessment plan.

¹² The logic in compiling these tables is the same followed in some process definition notation approaches like the ETVX (Entry-Task-Validation-eXit) [10] one.

The context information of the organisational unit should include the business goals and the departments goals, quantified if possible. Based on the stated goals, the selection of the most relevant processes and results to be assessed should be done by the software department responsables with the support of the assessment team.

During the preparation phase carried out in the two software departments, the selection of ENABLER processes and SPICE software processes was based on not very accurate context information due to the organisations immaturity. The organisational level business goals were clear for the two departments but the software department goals were not clearly defined and stated in one of them and they were too general in the other. Any of the organisations assessed had indicators to follow the accomplishment of business goals and no measures taken know the stakeholder's perception.

The department's goal statements showed a very weak alignment between top-level business goals and software department goals.

- **EXECUTION PHASE.** One of the most relevant tasks of the execution phase is the *Collect Data* task. The interviews carried out to complete this task were based on a set of questions prepared for each participant. A summary of what happen during the interviews, by type of processes and results assessed, is:
 - *ENABLER processes.* A subset of ENABLER processes (selected during the preparation phase) was assessed in each department. The questions related to each process were of several detailed levels, each level corresponding to different degrees of maturity of the process in the organisation. In the two assessments carried out, only the questions related to the first level could be asked due to the immaturity of the processes in the departments assessed. Many questions were left unanswered.
 - *SPICE processes.* The set of SPICE processes selected as key processes during the preparation phase did vary during the interviews. That is, the SPICE processes considered key processes according to the information provided by the department responsables was found not to be adequate during the interviews and in one assessment, two other SPICE processes had to be added to the assessment.
 - *Results.* First-level questions showed that the results were not measured in a formal way.

The execution phase showed that a higher level of maturity is required in the departments in order to fully check the assessment method and the underlying model and, it confirmed the weak alignment between the business management system and the software department management system.

- **REPORTING PHASE & FOLLOW-UP PHASES.** As a trial result each organisation received a report of three parts:
 - A list of the main strengths and areas to be improved;
 - For each assessed process, a complete list of the strengths and areas for improvement found and, the profiles of each process;
 - A list of recommendations for starting an improvement initiative.

The list of strengths for the ENABLER processes in both organisations was very poor. The rating of ENABLER processes was complicated. In addition to this, the EFQM/SPICE assessment based on the model in its 1998 version does not provide an EFQM score as assessment result. Most of the organisations, including the organisations assessed that was already involved in an EFQM improvement initiative are interested in knowing the EFQM score of the assessment.

The areas of improvement provided for the result criteria were poor. The EFQM/SPICE model provides a list of areas and attributes adapted to Software Intensive Organisations (SIO). However, a list of measures linked to each attribute that can be used as reference for a SIO would help.

The SIOs involved in an EFQM improvement initiative were not involved at all in the initiative even if the department responsables recognise as fundamental the contribution of the SIO for the results of the organisation.

Some positive aspects were observed in the assessments:

- The ability to refine or redefine the initially stated business goals with the assessment results;
- The assessments highlighted the extent of the role software has in an organisation;
- The assessments enabled organisational managers to get a better understanding of how software is developed;
- The assessment highlighted gaps between organisational business management and the software management increasing awareness of the need to manage software based on organisational needs;
- The SPICE like process assessment approach made it easier for software practitioners to interpret the EFQM criteria;
- The SPICE like process structure of the EFQM/SPICE integrated model provides an interpretation of EFQM for SIOs that is very useful to provide strengths and improvement areas for the software organisation or department assessed.

5. Lessons learned from 1998 trials

The experience ESI gained during 1998 EFQM/SPICE trials can be summarised in the following points that represent the most relevant input for the 1999 updating of the Integrated Model:

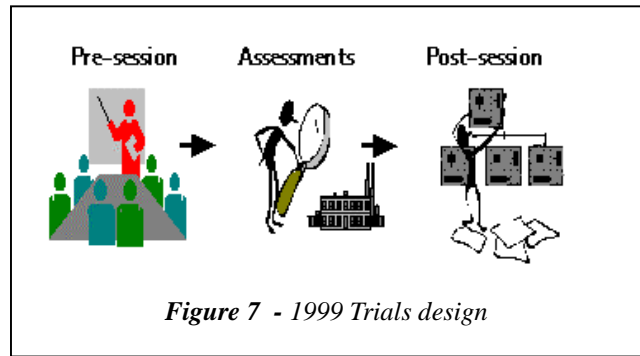
- A guided EFQM self-assessment will help organisations to define better basic pillars such as business goals, policy and strategy and key processes of the EFQM/SPICE Integrated Model;
- The EFQM/SPICE Integrated Scoring System should be compatible with the EFQM scoring system;
- The Result criteria part of the EFQM/SPICE Integrated Model should be enriched in order to provide richer support to SIOs willing to implement the EFQM/SPICE Integrated Model;
- Some organisational processes related with organisational issues such as *Leadership* or *Policy and Strategy* are difficult to assess at the department level when the full organisation is not involved in a TQM improvement initiative. The organisation top management commitment and participation is therefore essential for the complete success of this type of initiative.

The new updated model covers all the above points including also its *restyling* according to the new EFQM Excellence Model ("Version 2000").

6. Trials 1999

Based on the previously mentioned 1998 trials outcome, ESI has improved and redesigned the trial scheme for 1999. Figure 7 shows the three phases:

- **Pre-session**¹³: a tutorial for SIO managers is provided. Its aim is to present benefits and ways to align SPI into the wider context of TQM. An Internet on-line version of the tutorial is now being developed to support the pre-session.
- **Assessments**¹⁴: based on
 - A two-day self-assessment with EFQM criteria is carried out by SIO managers appropriately guided and supported on the initial attempt and,
 - A one-day mini SPICE assessment is performed by ESI consultants on two or three software processes identified as key processes.
- **Post-session**: identification and prioritisation of the improvement areas are derived from the assessments.



A 1999 joint trial with top managers of four SIOs has been planned. These trials have been organised in collaboration with EUSKALIT which is the EFQM representative in the Basque Country.

7. Conclusions & Prospects

Achieving business excellence in SPI is the main goal of Software Intensive Organisations. But traditional SPI models are not TQM oriented, while on the other side TQM models do not provide any help in defining key processes for a SIO and how to improve them. The European Software Institute has provided the EFQM/SPICE Integrated Model, a unique model that combines the strengths of two well-known models: EFQM and SPICE. This model can be used for assessment or for business improvement and planning purposes. A brief description of the three-tier architecture is presented as well as its 17 bottom-level tasks.

Main outcome from the 1998 trials are presented, classified by assessment phase, stressing the lessons learned. These are now input for the 1999 updating of the Integrated Model. This has

¹³ More than 50 managers of 48 different companies, among SIOs of the Basque Country and members of ESI and SPIN-SPAIN, attended a tutorial session that took place in Bilbao (Spain) last July. The feedback collected about the model was satisfactory, as inferable from the following table (based on a 40 questionnaires answered):

| Aspect \ % | EXCELLENT | HIGH | GOOD | ENOUGH | NOT SUFFICIENT | NOT AT ALL |
|----------------------------------|-----------|------|------|--------|----------------|------------|
| Interest created by the model | 21 | 31 | 33 | 15 | 0 | 0 |
| Usefulness of the model | 15 | 25 | 45 | 15 | 0 | 0 |
| Innovation provided by the model | 8 | 47 | 34 | 11 | 0 | 0 |

¹⁴ Four SIOs among the interested ones that attended the seminar have been selected in order to carry out the Assessment and Post-session phases of the 1999 Trials. The Trial results will be available for the end of the year 1999.

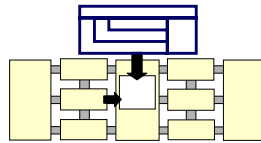
been very useful to ESI also in order to redesign and better organise 1999 trials, as demonstrated from a recent survey on the perception some SIOs have of the interest, utility and innovation provided by the Integrated Model for applications inside their organisations. After the trial phase, ESI is planning to promote the EFQM/SPICE Integrated Model application on the European level and different initiatives are being put in place to achieve this. Interest in participating in such activities is welcome.

References

- [1] BØTTCHER P., *Total Quality Management and the Capability Maturity Model (CMM): What's the difference?*, Proceedings of the IRIS20 Conference, Hankø Fjordhotel, Norway, August 9-12, 1997, <http://www.ifi.uio.no/iris20/proceedings/17.htm>
- [2] CONTI T., *Building Total Quality: a guide for Management*, Chapman & Hall, 1993
- [3] COVEY S.R., *Principles of Total Quality (Part 4)*, Modern Office Technology, Vol. 37 No. 2, February 1992, p.10
- [4] EFQM, *Self-Assessment 1997. Guidelines for Companies*, European Foundation for Quality Management, 1996
- [5] EFQM, *The EFQM Excellence Model – Improved Model*, European Foundation of Quality Management, 1999
- [6] GARCÍA A.B., BENGURIA G., OSTOLAZA E., ESCALANTE M.L. & SATRIANI G., *EFQM/SPICE Integrated Model V 1.0*, European Software Institute, ESI-1999-TR-003, February 1999
- [7] ISO/IEC TR 15504 Software Process Assessment's Parts 1-9, Technical Report type 2, 1998
- [8] KANJI G.K. & ASHER M., *100 Methods for Total Quality Management*, Sage Publications, 1996
- [9] PAULK M., TQM and CMM: Software CMM Q&A #4, April 7th 1997, <http://www.sei.cmu.edu/activities/cmm/docs/q-and-a.4.html>
- [10] RADICE, R. A. & PHILLIPS, R.W., *Software Engineering: An Industrial Approach*, Prentice-Hall, 1988
- [11] SAIEDIAN H. & KUZARA R., *SEI Capability Maturity Model's Impact on Contractors*, IEEE Computer, IEEE Computer Society, Vol. 28 No. 1, January 1995, pp. 16-26
- [12] SILVER B., *TQM and the SEI Capability Maturity Model*, Software Quality World, Vol.4 No.2, December 1992
- [13] ZORRIKETA I., GARCÍA A.B., BENGURIA G. & ESCALANTE M.L., *EFQM/SPICE Assessment Method Description*, European Software Institute, ESI-1999-TR-004, February 1999

Achieving Business Excellence in SPI:

Applying the EFQM/SPICE Integrated Model in Industry



L. Buglione & E. Ostolaza

EUROPEAN SOFTWARE INSTITUTE

E-mail: {luigi, elixabete}@esi.es

Agenda

- Introduction
- The EFQM/SPICE Integrated Model 1998 version
- The Assessment Method
- Trials 1998
- Lessons learned from Trials 1998
- Trials 1999
- Conclusions and Prospects

Total Quality Management (TQM)

- Covers all the aspects of the organisation
- Ensures a continuous alignment of improvements towards current real business objectives and needs
- Process improvement objectives are derived from organisational business objectives, thus focusing the improvement only on those areas related with those business objectives of the organisation

TQM Models

A descriptive representation of the elements and their interrelationships to take into account and to facilitate the understanding, reasoning, simulation, ...of the way towards total quality

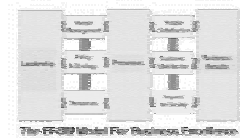
Most diffused TQM models are:



Deming



Malcolm Baldrige



EFQM

Problems when implementing TQM

TQM models provide a mechanism to define the policy and strategy for the organisation,

but

do not provide much support when identifying and defining the processes of the organisation that have most impact in business results and that keep aligned to the organisational policy and strategy

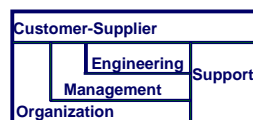
Software Process Improvement (SPI) models

- are oriented to facilitate the identification, definition, management, control and improvement of the software processes but not ensuring the alignment of processes to the organisational policy and strategy
- describe the processes involved in the development of software, providing for each of them the purpose, the basic activities, inputs, outputs,...that facilitates the implementation of the processes within the organisation

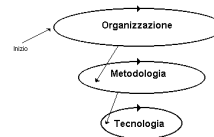
Most diffused SPI models are:



SW-CMM



SPICE



Bootstrap

SPI Models common features

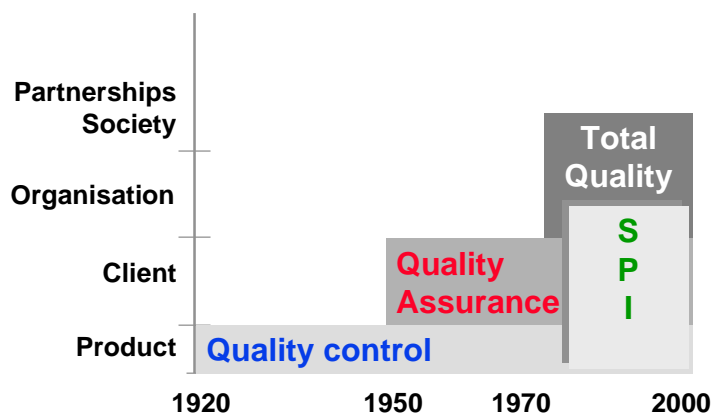
- Identify the set of processes that cover the model
- Describe the basic aspects that need to be accomplished when implementing the processes
- Describe the degree or level of discipline with which implement each of the processes

Problems when implementing SPI

- Reduced view of the improvement areas
- SPI objectives could not be completely aligned with the current policy and strategy and business objectives of the organisation



Steps in the evolution of quality



Current situation

- Quality in software industry reduces to SPI
- Few application and adaptation of EFQM by software industry
- It results too difficult that an organisation includes software related processes as key processes for business management

As a consequence...   **TQM**

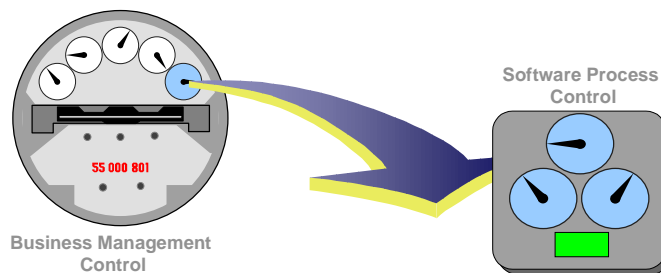
- SIO organisations approach quality only under the perspective of SPI
- Software departments of organisations under TQM initiatives get isolated and not involved dynamically on them

Why to integrate TQM and SPI?

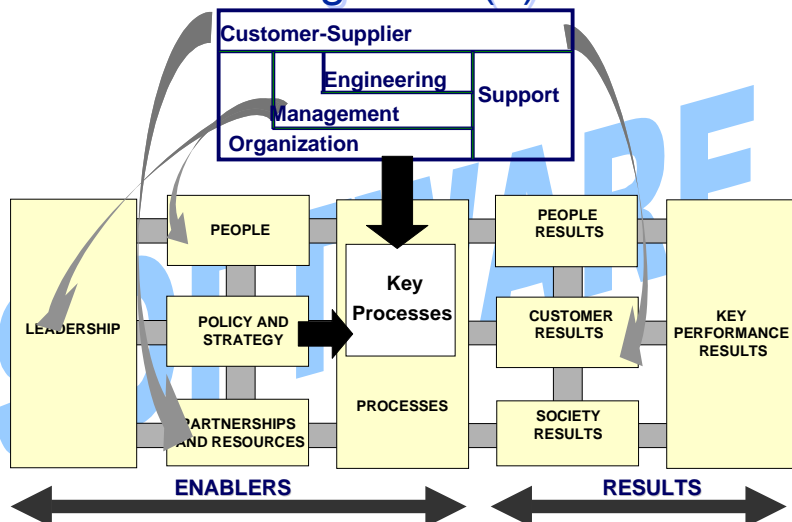
- **Integrate** software engineering and production processes in the business management
- **Deploy** business objectives of the organisation into software process improvement objectives that could be directly managed by software process managers
- **Provide** SIO organisations a more complete view of the areas where quality can be improved
- **Provide** to those SIO organisations approaching TQM the appropriate support to facilitate organisations the involvement of their software departments within their TQM initiatives

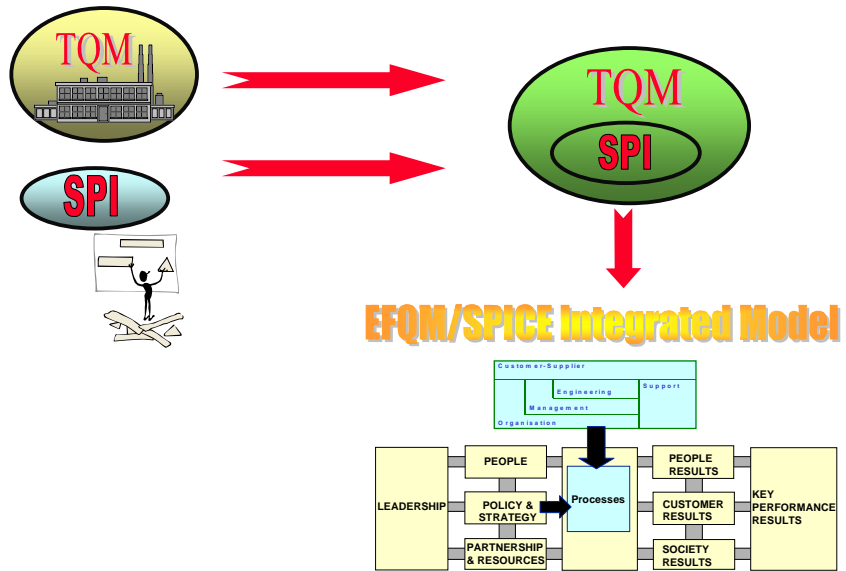
Result of the integration (I)

- Business improvement will cover improvements of the software processes
- Ensure that any improvement on software processes will contribute to business improvement

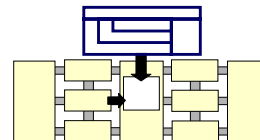


Result of the integration (II)





EFQM/SPICE Model



Components

- **EFQM Excellence Model (2000 Version)**
- **ISO/IEC 15504 TR-2** [aka as **SPICE** - **S**oftware **P**rocess **I**mprovement and **C**apability **d**etermination]

Objective

Provide a unique model covering software process improvement under a framework of Total Quality Management

Scope

Full coverage of the organisation business including those activities related to software development and maintenance

EFQM/SPICE Model Audience

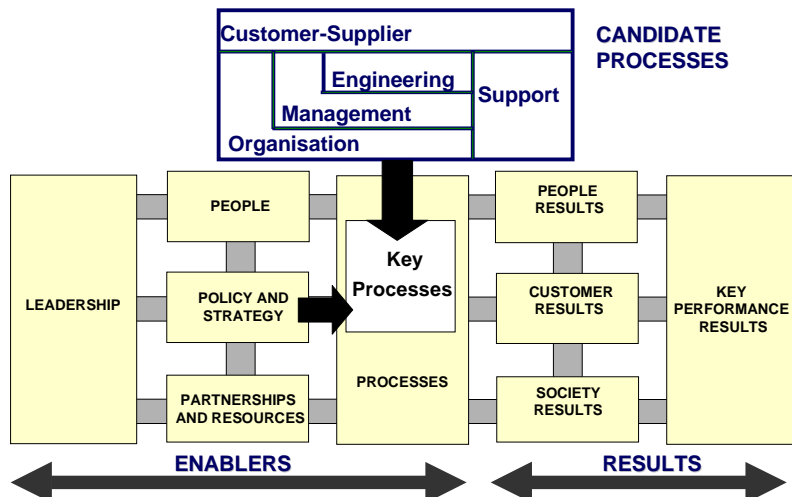
Software Intensive Organisations (SIO)

- Organisations whose main objective is software development and selling

Or

- Software Departments of:
 - Organisations that develop software as integrating part of its final products
- Or
- Organisations that develop software for internal use to achieve better business results or whose software department can be qualified as an independent organisational unit.

General structure of the model

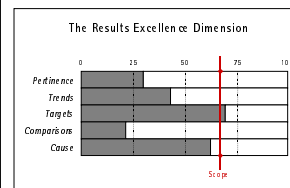
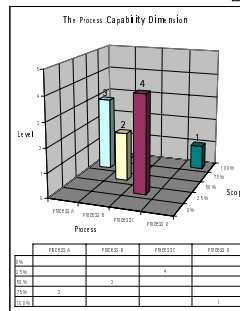
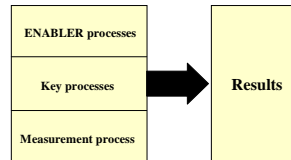


EFQM/SPICE Integrated Model Dimensions

The Integrated Model, like SPICE, has 2 dimensions:

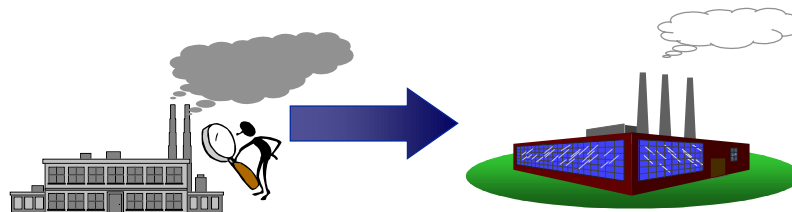
Processes & Results

- process side
- results side
- **Capability**
 - process capability
 - results excellence



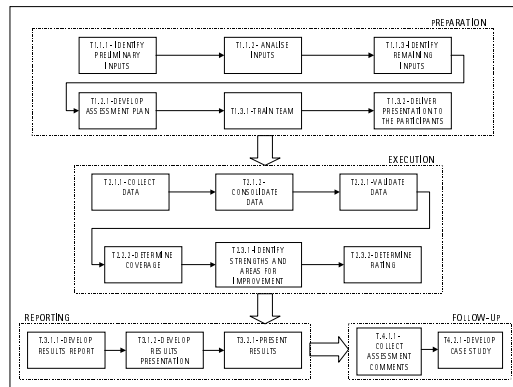
Model Applications

- **Self-assessment to measure the progress towards business excellence.**
- **Reference model for business management**



EFQM/SPICE Assessment Method

The Assessment Method defines four first-phases, with a list of activities and tasks to be performed in each phase, for a total of **17** bottom-level activities:



Trials 1998

Main outlines from 1998 Trials that involved **2** Software departments of Large organisations:

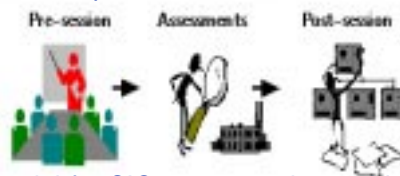
- **Preparation Phase:** *The department's goal statements showed a very weak alignment between top-level business goals and software department goals*
- **Execution Phase:** *a higher level of maturity in the departments is required in order to fully check the assessment method and the underlying model and, it confirmed the weak alignment between the business management system and the software department management system*
- **Reporting & Follow-up Phases:** *The SIOs involved in an EFQM improvement initiative were not involved at all in the initiative even if the department responsables recognise as fundamental the contribution of the SIO for the results of the organisation*

Lessons Learned from 1998 Trials

- **Guided EFQM self-assessment help to define basic organisational pillars (BGs, policy, strategy, KPs)**
- **More detail of Results (GQM-based)**
- **More detailed Rating scheme (EFQM compatible)**
- **Essentiality of Top management commitment**

Trials 1999

Based on 1998 Trials, ESI has improved and redesigned the Trial scheme for 1999. Three phases are foreseen:



- **Pre-session:** a tutorial for SIO managers is provided in order to present benefits and ways to align SPI into the wider context of TQM.
- **Assessments:** 2-days self-assessment of EFQM criteria and 1-day A SPICE mini-assessment on two or three software processes identified as key processes.
- **Post-session:** identification and prioritisation of the improvement areas are derived from the assessments

Conclusions & Prospects

- **Process Oriented - best from TQM and SPI**
- **Adapted to Software Intensive Organisations (*SIO*)**
- **More detail of Results (*GQM-based*)**
- **More detailed Rating scheme (*EFQM compatible*)**
- **New upcoming version 2.0 (*December 1999*)**

Improving software documentation: a customer oriented approach. The results of the ESSI PIE DOCPROVE project

Pietro Moro, Optec srl
Antonio Cicu, MetriQs srl

Abstract

The DOCPROVE project [11] is an ESSI PIE (Process Improvement Experiment) funded by the European Commission and undertaken by Optec srl, an Italian small company operating in the development and commercialisation of optical and opto-electronic custom systems. The paper covers the following main points of the improvement project: the essentials of the approach (driven by business goals), including the **ami** and GQM approach for metrics; the details of the application of SPICE process models to software documentation (in the framework of a SPICE trial), with emphasis to the adopted systemic approach; the effectiveness and ease of use of the adopted templates and tools; the results of the measurements and of the SPICE final assessment; the lessons learned regarding what has been really improved versus sustained costs, also from the point of view of motivation, training and reaction of people; and finally, an evaluation of experience replicability inside and outside Optec.

*Pietro Moro, Ing., Optec srl,
Via Canova 10, I-20017 Rho (MI) Italy
ph: +39 02 9350 1157, fax: +39 02 9350 0207
e-mail: moro.optec@agora.stm.it
website: <http://www.optec-srl.com>*

*Antonio Cicu, Ing., MetriQs srl,
Via don Gnocchi 33, I-20148 Milano Italy
ph/fax: +39 02 4870 8691
e-mail: acicu@metriqs.com
website: <http://www.metriqs.com>*

1. Introduction: the Optec's business and PIE project context

The **mission** of Optec is the development of optical and opto-electronic imaging systems. Optec is competitive on the market where critical and non-conventional systems are required. Figure 1 illustrates the market segments typically served by Optec.

Optec's **business process strategy** is supported by:

- a constant monitoring of the trends of **optical systems market**,
- a constant monitoring of the **concerned technologies** (for example: electronics components, and embedded software),
- a sustained interaction with the Customer.

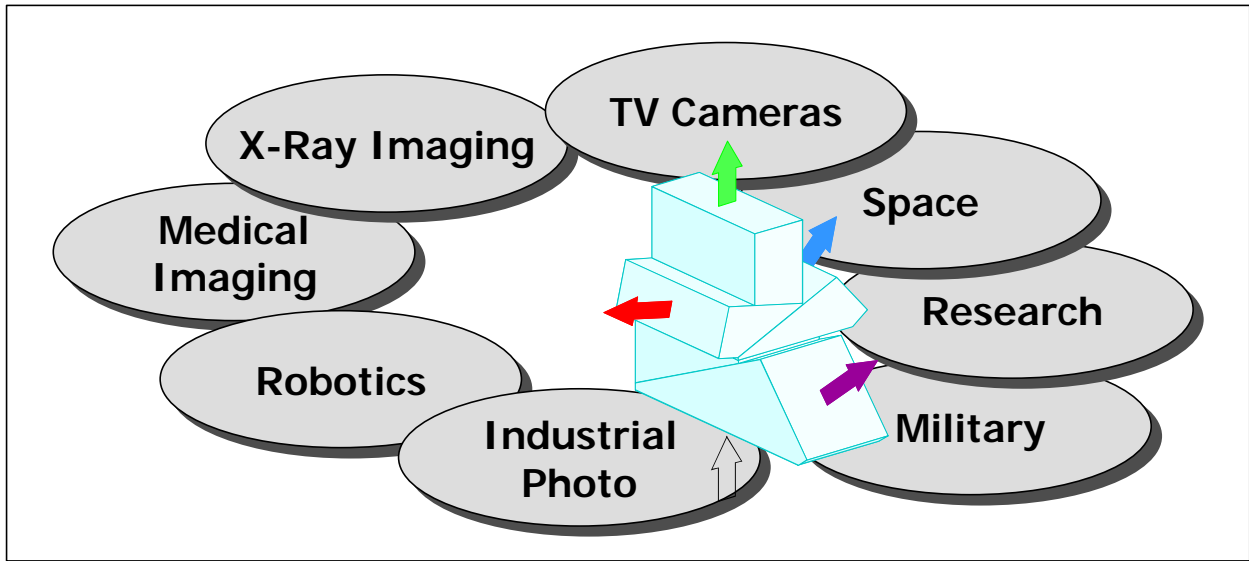


Figure 1 – The mission of Optec

Optec is able to assist the Customer in the conception of a **new system** and/or **integrated solution**, whenever an optical image has to be handled, and to supply the Customer with the complete product, co-operating as system designer and system integrator with other partner Companies, through the business cycle shown in Figure 2.

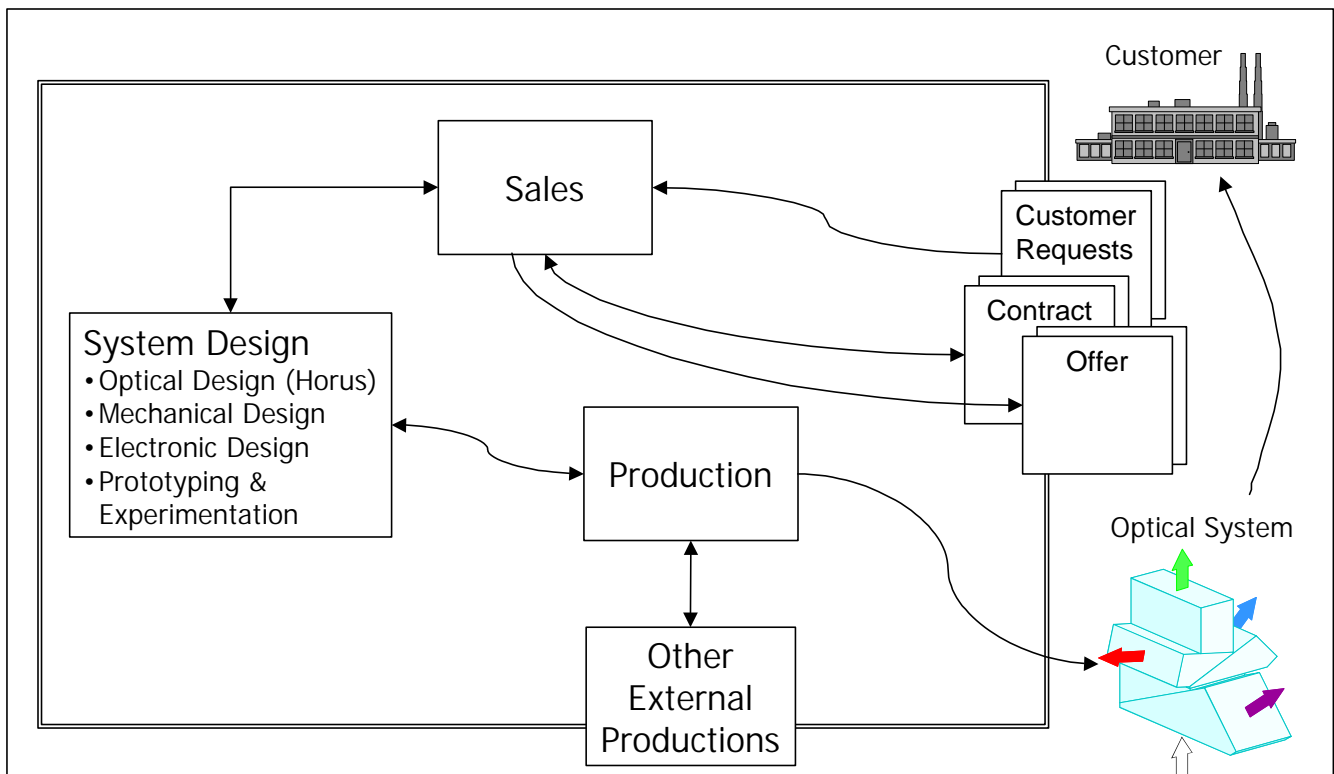


Figure 2 – The business cycle

A deep interaction among Optec departments is always necessary, as follows:

- the feasibility study and offer require a lot of optical design;
- the optical layout has to be 'dressed' by mechanical components, and compromises have to be kept into consideration;
- production needs the interaction with the optical and mechanical designers to prepare assembly tools and to define the testing procedures and tools;
- external suppliers have to be co-ordinated, and their products to be controlled;
- feedback from each department, from the external manufacturers, and from the customer is necessary for a better design and for product engineering.

The system design is **driven by the optical design**, that is carried out with the aid of proprietary software (the program Horus, so named from the Egyptian God of Sun).

And optical design support software is the area where the improvement action has been planned with the software process improvement project described in this paper.

The DOCPROVE PIE has been based on a Baseline Project (Figure 3), which had the objective of making Horus to evolve from the current release to a new release.

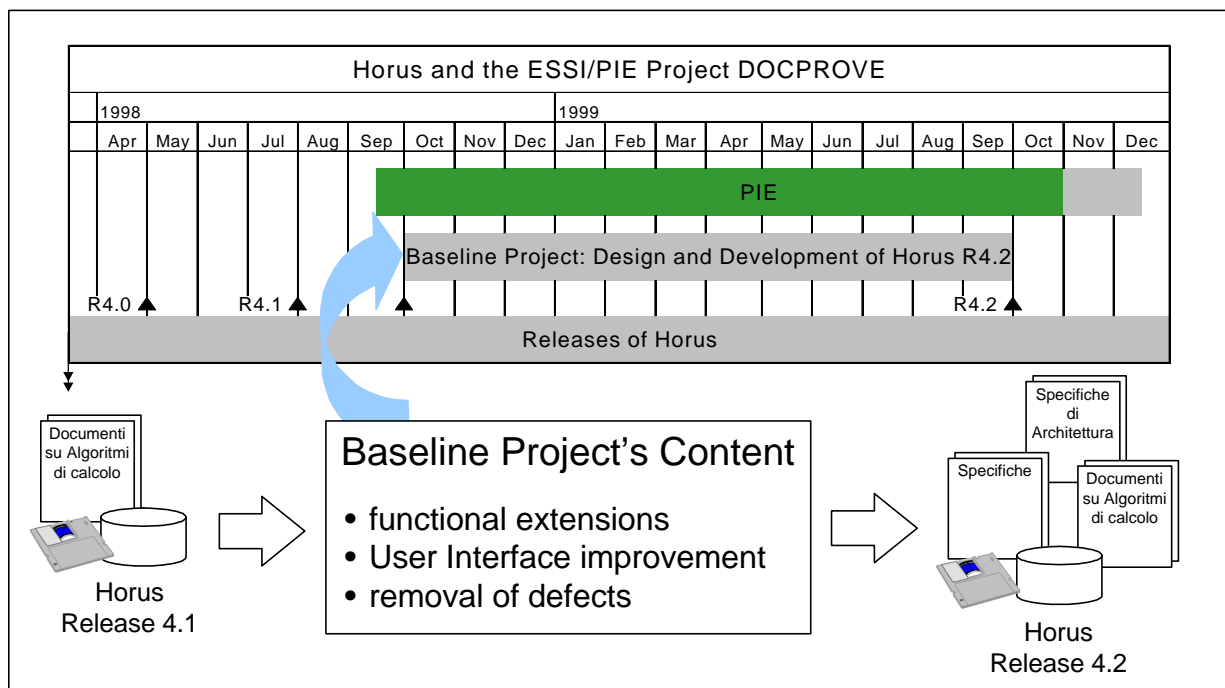


Figure 3 – The baseline project

The objectives of the new release (R4.2) of Horus were:

- add new design features;
- port the tool to faster machines, making the design activity more efficient;
- make the task of the optical designer easier, thanks to enhanced User Interface;
- remove known defects.

The tool Horus has a strategic importance in the organisation, because it is the design tool daily used to develop (or upgrade) optical systems. It embeds a lot of know-how of mathematics and optical science, and it undergoes a yearly upgrade activity as required by specific optical design requirements.

The Baseline project did start in October 1998: now (November 1999) the Baseline project has been concluded, and the PIE is in the final reporting phase.

During the Baseline project a new process of documentation was experimented, applied to the parts of Horus being worked.

Also documentation processes of Optec at system level were involved in the Baseline project, and the documentation of optical systems at system level was included in the improvement experiment, in order to provide to software people a set of software requirements as solid and stable as possible.

2. The objectives of the DOCPROVE project, and the starting scenario.

The **business goals** of Optec can be synthesised as follows.

- Goal 1** Design and production of innovative, not conventional systems, in the field of image acquisition and image processing.
- Goal 2** Increase its presence in important market areas, such as in-line image acquisition and processing, in-line quality inspections, medical equipments, R&D (e.g. nuclear, space, energy, environment fields), where there is the need of a complete cycle of services (user needs analysis, conception of the solution, design, production, installation and support) regarding the above optical systems (the capability of supporting a complete project cycle being a distinguishing factor for Optec).
- Goal 3** Extend the range of Optec products applicability through ease of customisation and timeliness of solution provision.
- Goal 4** Reduce times and costs for solution provision; increase margins.

A program for pursuing the above **business goals** originates a set of **needs**, that have to be satisfied in order to achieve the goals. For each need the following list provides also the mapping to pursued goals:

- Need 1** Improve the ability of system and software analysis and specification (pursuing goals 1, 2, 4).
- Need 2** Improve the ability of evaluating (through the available documentation) the suitability of external application software candidate to be integrated into Optec systems (either to be acquired, or to be subcontracted) (pursuing goals 1, 2, 3).
- Need 3** Availability of efficient and up-to-date optical design tools (including the ability of implementing timely changes, as much as possible by less experienced technicians) (pursuing goals 3, 4).
- Need 4** Formalise and document the competence and expertise (related to optical systems) embedded into Horus design tools (such a competence has to be an accessible and reusable intellectual property of the Company, and must not remain in the mind of the tool developers). (Asset to be mastered by the Company, control of investments) (pursuing goal 3, 4).
- Need 5** Consolidate, improve the ability to integrate outside subsystems in Optec systems (pursuing goal 2).
- Need 6** Stronger ability to coordinate activities (internally, and externally with business partners) (pursuing goal 4).

The following points were the **yardsticks (YSn)** (measured reference points) by which it was possible to measure the progress towards the achievement of the objectives and results:

- YS 1** the definition of PIE monitoring approach, including ami goal tree, the measurement plan, and metrics to be used (at month 1.5);
- YS 2** the definition of baseline project documentation process (at month 2.5);
- YS 3, YS 4, YS 5** the Technical Review Reports (including the results of usage of methodology and tool, and the measures) (at months 7, 10, 13);
- YS6** the Operation Phase Results Report (at month 15).

A CMM-based mini-assessment performed in the software department during early 1996 (driven by CMM [6] inspired criteria and by the CMM Maturity questionnaire) pointed up that main weaknesses were in the documentation process. While people were used to produce their own documentation regarding their own activities, no standards were in use, though attempts had been done in the past in order to standardise the process.

The motivation for the PIE came from the management awareness that documentation practices improvements are a prior step to satisfy the needs mentioned above, and specifically to get:

- higher maintainability and extensibility of Horus’ future releases (**business relevance and benefits**: shorter time-to-market, new optical design capabilities);
- higher ability to evaluate (through the available documentation) the suitability of custom application software components to be integrated into Optec’s commercial systems (**business relevance and benefits**: higher competitiveness).

Among the mentioned Optec’s business goals, priority was given to Goal 3 and Goal 4 as driving criteria for the selection of improvement actions.

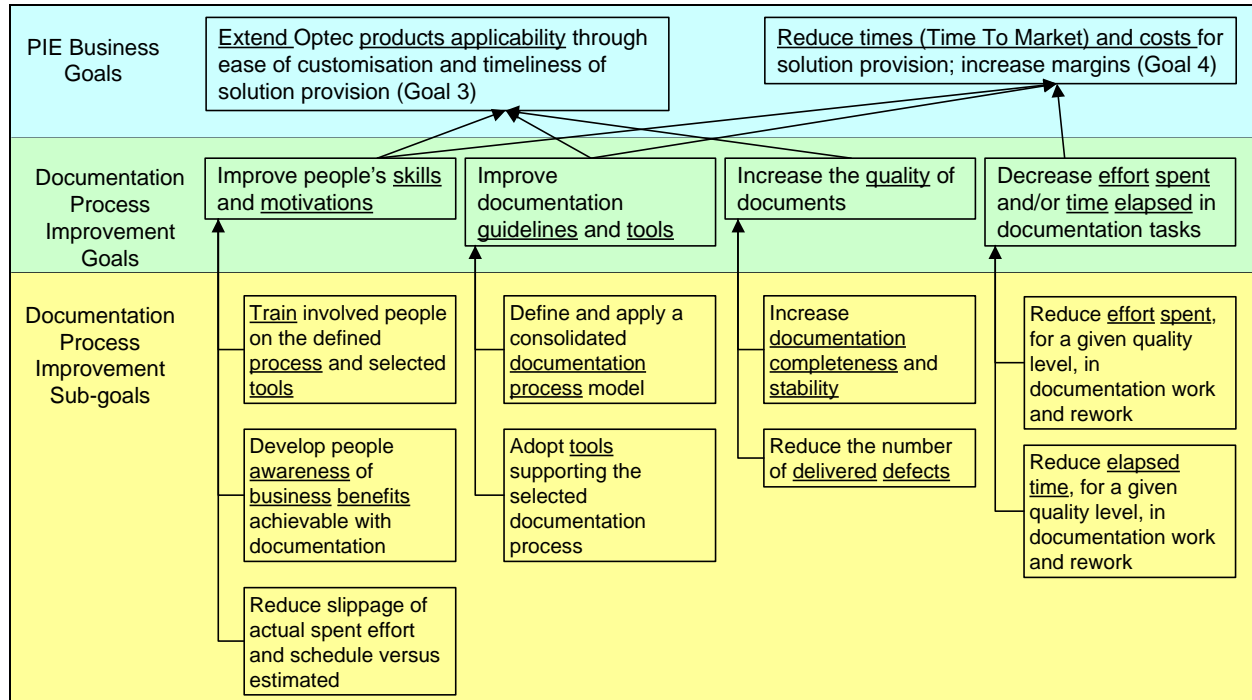


Figure 4 – The PIE Goals Tree

Being the selected **overall objective** of the experiment the **improvement of the software documentation practices**, such a major objective was concretised in the following **specific sub-objectives** (for each sub-objective the information is provided about **which measures** have been selected, with the help of **ami method** (Figure 4) [5], in order to control the level of achievement and the impact on the business goals, and in order to verify to what extent the improvement goals were achieved):

- to **improve the skills and motivations of the people** involved in software specification and documentation tasks, and in documentation review tasks (people will have to improve their knowledge of why and when to produce software documents, of why and when to review them and control their quality, of who is the audience of a given document) (measures: evaluation of documentation methods competence levels, questionnaires for evaluating the employee satisfaction);
- to **improve the documentation guidelines and documentation tools**, as well as the documentation control policies and tools (people will have to improve their knowledge of what and how to document given software product aspects, of which tools can help for saving time and for increasing quality) (measures: evaluation of the documentation process approach in terms of completeness and compliance to the international standards);
- to **increase the quality of documents**, at parity of spent effort and elapsed time (measures: completeness, consistency, non-ambiguousness of produced requirements documents; defect data and defect density for defects found in the produced documentation);

- to **decrease spent effort and/or elapsed time** in documentation tasks, at parity of the other factors (respectively time elapsed and quality, or effort spent and quality) (i.e. to improve the documentation process efficiency) (measures: documentation time efficiency and documentation cost efficiency).

The life cycle adopted in the experiment is basically inspired from ISO/IEC 12207 [1] and ISO/IEC 15504 [7] standards (see [17]).

The addressed process activities were (Figures 5 and 6):

- **document preparation activities.** Using proper document preparation guidelines (for selected critical parts of the User manual, and for the technical documentation, with priority given to the user needs specifications, the system and software requirements specifications, the user manual). The guidelines had to provide the suitable help to write user needs and product specifications driven by the selected main business objectives;
- **documentation quality verification.** Requirements quality verification was driven by the document model described in the selected templates, exploiting the requirements structure (requirement description + requirements attributes) and traceability, in order to make more precise and cheaper the verification, including the adequacy to the user needs, and the aspects of completeness, consistency, feasibility and testability of specified requirements;
- **problem reporting, document change request and control, and document rework.** These activities were performed in the cases where the reviews detected any non conformity with the adopted templates or with other requirements.
- **data collection and metrics calculation.** Data were collected and metrics calculated during the experiment, as it was specified in the Detailed Plan. The approach to metrics is described in detail in the dissemination paper [17].

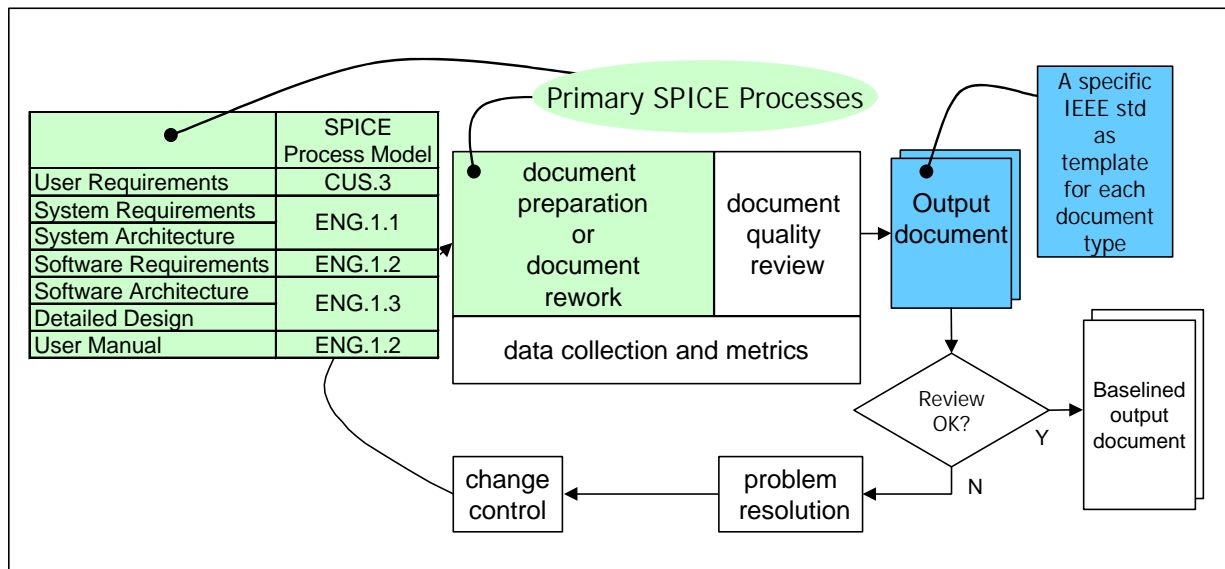


Figure 5 – Process activities: Primary processes (the object of the improvement)

Product documentation, due to the nature of the baseline project, was prepared following a system approach, to ensure that requirements to be allocated to software were correctly derived from system requirements. In order to better guarantee this system approach, *marketing people, and mechanical and optical designers did join software designers* in the initial specification activities, and in review works.

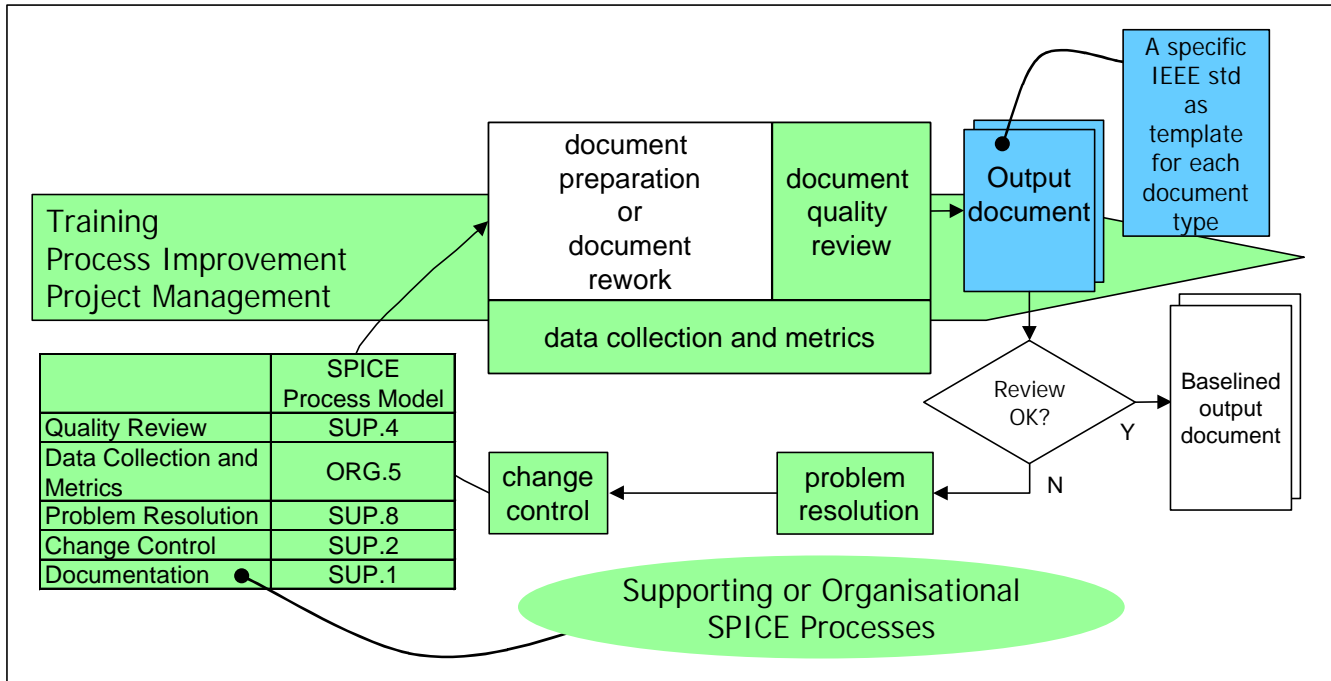


Figure 6 – Process activities: Supporting and Organisational processes

3. The description of the experiment.

The details of the approach adopted for the experiment were already published in the paper [17], illustrating the following aspects: life cycle model selected for a software project, rationale for the document types to be experimented, IEEE software engineering standards and guidelines selected to drive the definition of documents structure and content, rationale and calculation details for the metrics selected as improvement indicators.

3.1. Main processes affected by the experiment, and related methodologies and tools.

Two major groups of processes are affected by the experiment:

- a) primary documentation processes (Figure 5), which are the direct object of the DOCPROVE improvement action. The processes selected under this group are the ones which directly produce the different levels of specifications required in a system where a part of requirements is allocated to software, and the process (a support process, the Documentation process) which defines all the other documentation processes. The selected project life cycle is described in Figure 7, which shows also the project documents that were experimented in DOCPROVE: the Application Requirements (AR), System Requirements (SYSR), System Architecture (SYSA), Software Requirements (SWR), Software Architecture (SWA), Detailed Design (DD), User Manual (UM).

The corresponding primary documentation processes (again Figure 5), which produce the above documents, are: Requirements elicitation (for AR), System requirements analysis and design (for SYSR and SYSA), Software requirements analysis (for SWR and UM), Software design (for SWA and DD). All these processes belong to the SPICE Engineering process category, with the exception of the Requirement elicitation, which belongs to the SPICE Customer-Supplier process category.

There is another process that was directly affected by the improvement action: the Documentation process, which belongs to the SPICE Support process category (Figure 6).

Last but not least, one more process was significantly affected by the improvement action, even if such process was not included as an explicit objective of improvement: the Measurement process (belonging to the SPICE Organisation process category), thanks to the varied set of measures that were experimented as measures of improvement of the documentation processes mentioned above.

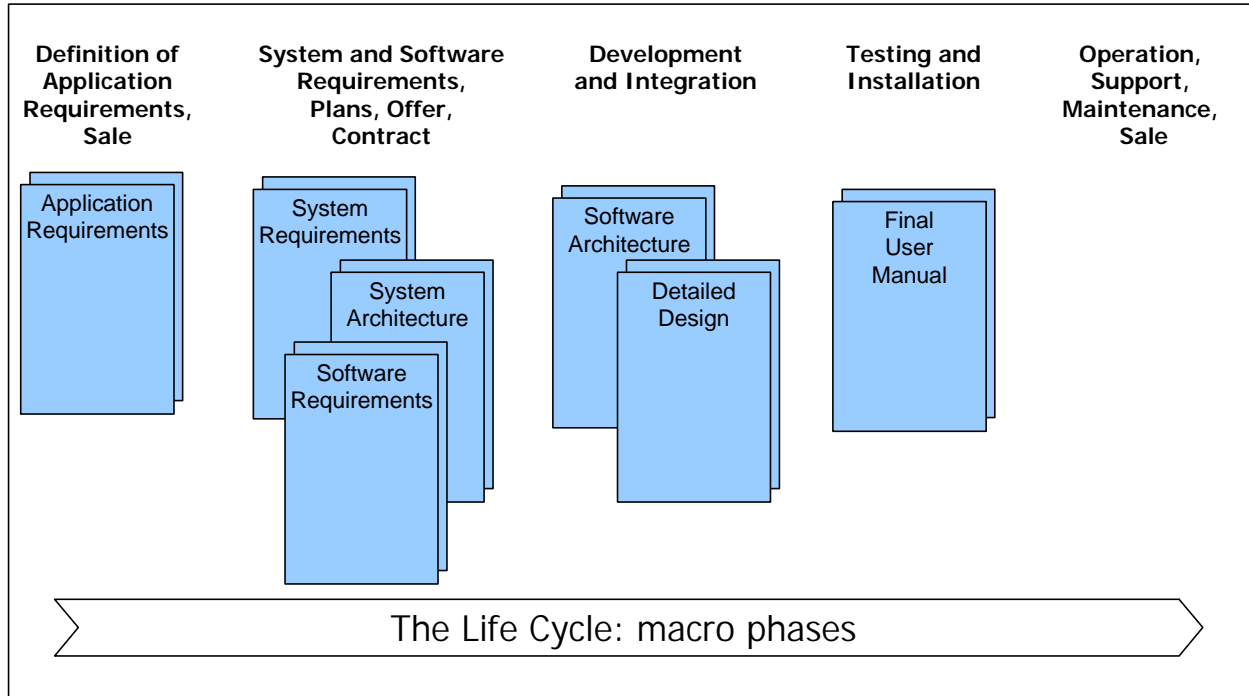


Figure 7 – Experimented documents

- b) supporting and organisational processes/activities. These are processes/activities which are invoked by the primary documentation processes in support of them, but which were not object of a specific improvement action under DOCPROVE (and for which, e.g., no measurements were produced).
 The processes included in this group are: Verification, Problem resolution and Configuration management (belonging to the SPICE Support process category); Project management (Management process category); Improvement process and Human resource management (Organisation process category) (Figure 6).
 The reason why these processes have been considered was just to profit of the guidance provided by ISO/IEC 15504 [7] for performing activities which are anyway to be performed within the project as necessary support activities, but with no need of a full application of the related SPICE models.

The cycle of activities performed when producing and reviewing/reworking a document in DOCPROVE is shown in both Figures 5 and 6.

Suitable IEEE standards or guidelines ([3], [4], [8], [12], [13], [14], [15], [16]), were selected as a guide to authors for the definition of the structure and content of documents.

Methodologies evaluated and introduced

The following table provides a description of the methodologies evaluated and used in the project, per each area of activities where a consolidated methodology was known.

Table 3.1-1 – Methodologies evaluated and used or introduced

| Activity or process area | Methodology evaluated and used or introduced (and evaluation of the degree of innovation: Degr. of Innov.) |
|---|--|
| Models for process definition and establishment | <p>The method adopted for process definition and establishment was to use as much as possible the process models described by ISO/IEC 12207 standard [1], with some practical details derived in part from the related IEEE addenda for 12207 ([3], [4], and in part from SPICE guidelines [7]. A schema of the considered process models has been presented at Vienna conference [17], and provided also in Figures 5 and 6.</p> <p>Reason of the choice: both ISO/IEC 12207 and 15504 provide clear, concise, consistent descriptions of the single process models. The work of deriving the defined process model for DOCPROVE has been straightforward. (Degr. of Innov.: High)</p> |
| Requirements and design specification | <p>A Requirement management methodology based on requirement elicitation and definition templates derived from IEEE guidelines ([12], [13], [14], [15], [16]), which encourage the use and control of requirements traceability and requirement attributes. Requirement definition and control activities have been modelled taking as reference the suitable SPICE models. Full reference to the adopted IEEE standards was published in the dissemination paper [17].</p> <p>Reason of the choice: the IEEE templates provide flexible, adaptable models for document structures, defined as a digest of industrial experiences, applicable also in small projects and in small teams. The use of traceability and requirement attributes supports the achievement of a higher completeness in review and rework. (Degr. of Innov.: High)</p> |
| Metrics | <p>The ami methodology [5] has been used for defining the goal tree (Figure 4). In the context of this methodology, the Goal/Question/Metrics (GQM) approach [10] has been used for defining the metrics to be used in the project. The rationale of the approach, and the results of the metrics definition were presented at the EOQ Vienna Conference [17]. Some criteria for measurements reporting were derived from the Measurement process model of SPICE [7].</p> <p>Reason of the choice: well documented methodologies, supporting an approach based on common and practical sense for linking the metrics to the selected business goals. The results were well understood and shared by Optec’s management. (Degr. of Innov.: High)</p> |
| Verification | <p>The guideline for conducting document quality reviews is derived from the SPICE Verification process model (SUP.4) [7], and from the IEEE standards supporting the Review activities ([8] and [9]).</p> <p>Reason of the choice: again the considered guidelines are a digest of long and diffused industrial experience, and convincing also for small companies. (Degr. of Innov.: Medium)</p> |
| Process assessment | <p>The selected guideline for conducting the planned process assessment is the one defined by the ISO/IEC 15504 standard (SPICE) for process assessment ([7], Parts 3, 4 and 5). A preliminary assessment (antecedent to the start of DOCPROVE) was based on CMM inspired criteria [6]. In addition to the CMM based mini assessment (1996), one more mini assessment, driven by SPICE guidelines, had been performed in 1997. Reason of the choice: SPICE models and guidelines are supported by accessible courses and documentation; the cultural investment appears to be a durable one, not open to risks of fast obsolescence. The model offers suggestions for defining the paths and actions for further improvements after the assessments. (Degr. of Innov.: High)</p> |

Tools evaluated and used

The following table provides an overview of the adopted tools, with our evaluation of the related degree of innovation. For the sake of completeness of the overview, the copies of the standards acquired in support of the project have been considered themselves as support tools.

Table 3.1-2 – Tools evaluated and used

| Activity or process area | Tools evaluated and used | | | | | | | | |
|---|--------------------------|---------------------|------------------------------|------------------------|------------------------|------------------------|---------------|----------|----------|
| | ProDoc | ISO ^{plus} | IEEE standards in PDF format | ISO/IEC 12207 standard | IEEE addenda for 12207 | ISO/IEC 15504 standard | Requisite Pro | MS Excel | MS Word |
| Models for process definition and establishment | x | x | | x | x | x | | | x |
| Requirements and design specification | x | | x | | | x | x | | x |
| Metrics | | | | | | x | | x | |
| Verification | | x | x | | | x | x | | |
| Process assessment | | | | | | x (Parts 3,4) | | | |
| <i>Degree of innovation supported by the tool</i> | <i>M</i> | <i>H</i> | <i>M</i> | <i>H</i> | <i>H</i> | <i>H</i> | <i>H</i> | <i>M</i> | <i>L</i> |

Legenda: L=Low; M=Medium; H=High

Reasons of choice of the above tools were the following: a) for the ISO/IEC standards, to make use of the reference international standards for software processes; b) for the IEEE standards, to make use of guidelines and templates resulting from consolidated and diffused industrial experience; c) for ProDoc, ISO^{plus}, RequisitePro, and Excel: after verification that the tools satisfied the needs at reasonable price.

3.2. The phases of the experiment

The experiment did span over four main phases:

1. Definition of the **initial status** and of the **improvement approach**; definition of metrics and their target for the experiment; definition of the preliminary PIE plan. Key deliverables: a document describing the PIE approach, and a paper [17] published in April 99, that illustrates the fundamentals of the approach.
2. **Experimentation.**
This phase included the following activities: definition of the documentation process, strategies, methods, tools and guidelines; installation of customised procedures for the documentation process and of the tools acquired; training of involved people; release of the detailed PIE plan; experimentation of new practices, and collection of data for deriving improvement indicators; exploitation of the collected data; comparison of results, correction of non-conformities. Key achievements: the documents to be experimented, the measure reports. Further major expected deliverables: the Final Assessment report, the Documentation guidelines for Optec.
3. **Result synthesis, final reporting.** Key deliverables: the ami case study, the Final Report.
4. **Dissemination** at international events. Key deliverables: 1st international dissemination event (achieved, [17]); this paper at QWE'99 Conference, and the related presentation [19].

The initial total cost estimate for the project was 105600 ECUs corresponding to an effort of 292 man-days and 58.5 consultant-days. A revision of the estimates made after 7.5 months of project produced a new estimate of the overall effort required for the experiment: 338 Optec man-days + 64.5 consultant-days (55 for consultant service, 9.5 as teacher for training).

4. The results of the experience

4.1. Technical impact

With reference to the **yardsticks** defined in Section 2 as measurable reference points, by which it is possible to measure the progress towards the achievement of the objectives and results, the following synthesis provides a view of the technical impact.

The problem areas, indicated in section 2 as object of improvement, have been technically tackled through a synergetic use of methods and tools, as indicated in Table 4.1-1.

Table 4.1-1 – Problem areas

| Problem area | How technically tackled |
|--|---|
| improve the skills and motivations of people | conduct the planned training sessions, covering adequately the adopted methods and tools |
| improve the documentation guidelines and documentation tools | define the documentation processes and the corresponding guidelines compliant as much as possible to consolidated and affirmed international standards |
| increase the quality of documents | experiment the adopted guidelines to support the production of the planned documents, and perform the selected quality measurements (documentation completeness and stability, delivered defects indicators) (see [17]) |
| decrease spent effort and/or elapsed time | perform the selected efficiency measurements, using the data collected during the experimentation activities (effort slippage, schedule slippage, effort efficiency, time efficiency) (see [17]) |

The two tables in section 3 “Methodologies evaluated and introduced” and “Tools evaluated and used” provide the information regarding the methodologies and tools which have been evaluated and adopted for performing the improvement experiment.

The **1st yardstick YS1 (Definition of the improvement approach)** has been achieved making the proper synergetic use of the of the ami [5] and GQM [10] methodologies, using also some elements of the measurement process of SPICE [7]. The fundamental points of the approach have been published and disseminated (see [17]).

The **2nd yardstick YS2 (Definition of the documentation process)** has been achieved [20] exploiting (as said in Table 3.1-1, first row) the process models and guidelines provided by the standards ISO/IEC 12207 [1] and ISO/IEC 15504 [7], integrated with the relevant IEEE guidelines, in order to produce the process descriptions, the document templates, and the documentation management procedures.

For each of the document types planned for the experiment, the DOCPROVE project team has verified that there is a corresponding process model in SPICE [7]. In the framework of DOCPROVE the processes, which have a technical specification document as direct output, have been called “primary documentation processes” (Figure 5).

Each “primary documentation process” has been defined, in DOCPROVE, making as much reference as possible to the process standard ISO/IEC 15504 [7], and using the following schema:

- Adopted process model
- Process purpose

- Responsibility
- Input
- Input criteria
- Process activities description
- Quality review of documents and of other process outcomes
- Output
- Baselineing of output
- Characteristics of process' Work Products
- Reference standards.

A remarkable result is the **explicit orientation to customer needs** of the document template (inspired from the IEEE standard 1362–1998 [12]) selected in support of the elicitation of **Application requirements**. This is considered one of the distinguishing aspects of the performed experiment: Optec management feels now to have acquired a useful guide for defining and agreeing the customer needs. The approach supported by the IEEE standard [12] requires that the analyst describes the following major points:

- the current product or process of the customer
- business goals of the customer
- which are the limitations or weaknesses of the current product or process constituting an impediment for the customer to achieve the business goals
- which are the modifications of the current product/process or new functions required to overcome the limitations
- the characteristics of the new product/process resulting from the installation of the modified or new functions.

The above schema stimulates the analyst to acquire and master the knowledge on customer's business processes: such knowledge allows the analysts and commercial people of the vendor to formulate the specifications of a new solution which matches as much as possible the customer problems and expectations. More, a good knowledge of customer's processes (i.e.: of its market) allows the vendor to even anticipate future needs of the customer and to be perceived from the customer as a real partner supporting its business.

The adopted general documentation process model (see the cycle in Figures 5 and 6) gives evidence of a set of support/organisation activities that are performed in order to control and rework correctly a document (review, problem reporting, change control, data collection and metrics), and to prepare and sustain the prerequisites for improving the documentation activities (training, project management, process improvement).

Again SPICE guidelines provide a process model for the above support/organisational activities. So, even if such activities are not the object of improvement in DOCPROVE (they are simply used in support of the primary documentation processes), the DOCPROVE team has considered useful to define a process also for them, taking inspiration from SPICE, with the aim of ensuring the best support conditions to the activities to be improved.

All the planned documentation experiments have been concluded. The resulting documents were submitted to technical reviews and reported in suitable **Technical review reports** (the **Yardsticks YS3, YS4, YS5**): the major outcomes resulting from the reviews have been used throughout this paper. The quality control activities performed through such technical reviews have been accurate, applying the review criteria documented in the relevant process descriptions [20]. A result is that Optec participants to the project affirm that the specification documents produced are **more complete** and **more stable** than the rather informal documentation used in the past for the system specification and design phases. The evidence is supported by the adherence of developed documents structure to the selected documentation models.

Change control of documents has not always been performed formally: in some cases the change requests were not formalised, but changes were performed directly.

The planned **data collection and metrics activities** have been performed. We now have the following results (previously not available):

- a set of metrics driven by project and organisational goals (supporting the control of product quality, and of process quality and efficiency)
- an initial practice of a data collection discipline
- an initial nucleus of historical data
- an initial set of measures against which to perform future comparisons.

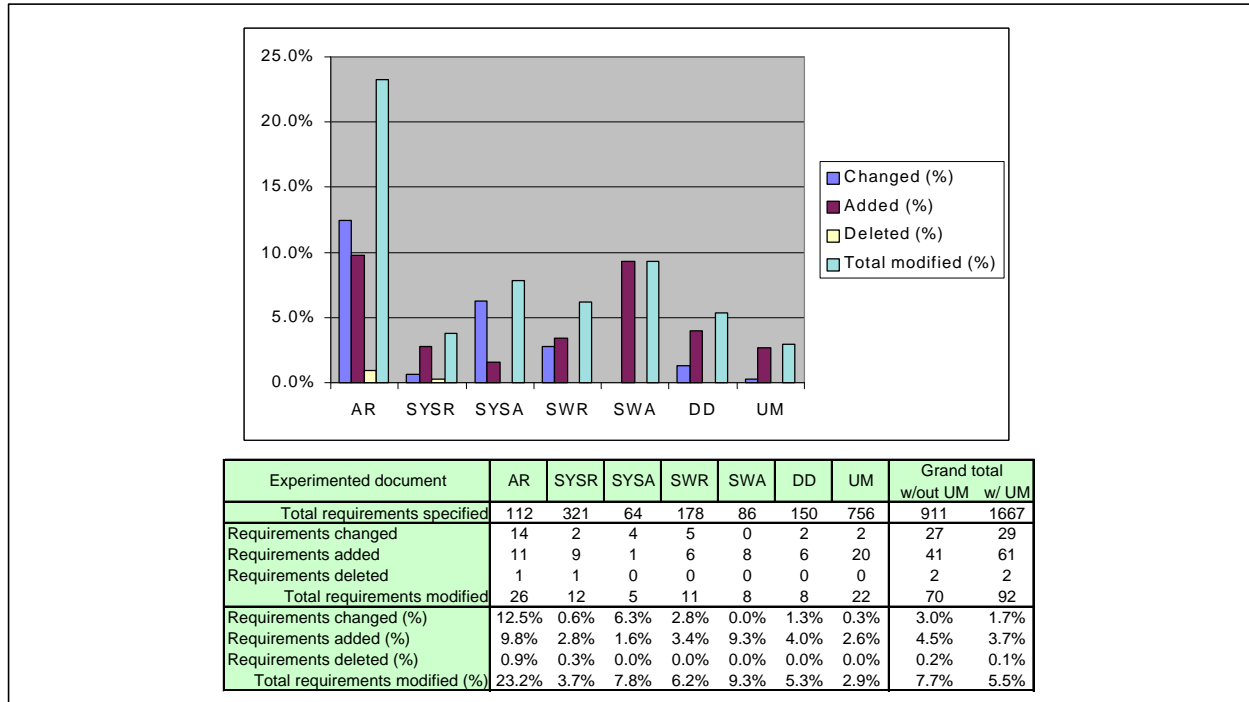


Figure 8 – Documentation completeness and stability indicators

Among the documentation quality metrics produced in DOCPROVE as planned, a set of **documentation completeness and stability indicators** has been produced, which can be used in the future in order to monitor future progresses in the requirements elicitation and specification capabilities (see Figure 8). The diagram shows that there has been some improvement, in terms of percentage of requirements modified, going from the initial phases of the experiment toward the conclusive phases. This could be a result of an increasing capability of mastering the topics suggested by the documentation templates.

The planned **process assessment** has been performed, with the purpose of evaluating the process capability levels of the processes that were to be improved.

The assessment has been performed in a way compliant to the SPICE guidelines (see [7] parts 3, 4, and 5). The assessment report provides a detailed analysis of the strengths and weaknesses for each assessed process, making precise references, as evidence of each finding, to the experimented guidelines and documents.

The results of the assessment are summarised in the **process profiles** shown in Figure 9. Figure 9 provides the results of the recent SPICE compliant assessment, performed on the processes performed with the support of the DOCPROVE project, and the results of an initial SPICE assessment made in July 1997. Basically it can be said that documentation processes capabilities are now at level 2, as far as the activities performed on the Baseline project are considered, and that a stabilisation of the achieved capability level is ready to be pursued making use, in the next projects, of the guidelines and experience accumulated in DOCPROVE.

From the point of view of **process assessment and improvement**, the following statements summarise the impact:

- **process modelling and description** has been supported in a consistent and practical way, as shown above, from the standards ISO/IEC 12207 (see [1], [3], [4]), and ISO/IEC TR 15504 [7]. The process descriptions produced in the document [20] are a relevant added value to Optec quality system, ready also to be used with ISO 9000:2000 which will require explicitly the description of the quality system by processes;
- **process assessment** was very well supported by the mentioned assessment guidelines ([7] parts 3, 4, and 5). Specifically, the guide available in [7] part 5 provides a set of process capability indicators which are helpful in

conducting a detailed review of process performance and in detecting and pointing out the major weaknesses. The assessment report produced with this guidance is a real asset for Optec, that contains practical and detailed indications for further improvement actions.

| Process | | Before the improvement project Process attributes | | | After the improvement project Process attributes | | |
|------------------------|--|--|------------------------|-------------------------|---|------------------------|-------------------------|
| SPICE process model ID | Process name | PA 1.1 | PA 2.1 | PA 2.2 | PA 1.1 | PA 2.1 | PA 2.2 |
| | | Process performance | Performance management | Work product management | Process performance | Performance management | Work product management |
| CUS.3 | Application Requirements document process | | | | | | |
| ENG.1.1 | System Requirements and System architecture document process | | | | | | |
| ENG.1.2 | Software Requirements document process | | | | | | |
| ENG.1.3 | Software Architecture and Detailed Design document process | | | | | | |
| ENG.1.2 | User Manual document process | | | | | | |
| SUP.1 | Documentation process | | | | | | |
| ORG.5 | Data collection and metrics (measurement) process | | | | | | |

LEGENDA

- not achieved (N) [white box]
- partially achieved (P) [light gray box]
- largely achieved (L) [medium gray box]
- fully achieved (F) [dark gray box]

Figure 9 – DOCPROVE project assessed process capability profiles

Thanks to the above positive impacts, in synthesis Optec’s management evaluates that the project has contributed to solve the following **major process problems**:

- a) **requirements instability.** Optec analysts and designers evaluate to be able now, thanks to the acquired documentation models and to the documentation quality metrics, to produce documents which are more complete, more stable, more customer oriented; and to possess experimented criteria for guiding technical reviews that help in timely detection of analysis and design defects and weaknesses;
- b) **the control of internal deliveries.** The technical review discipline experimented in the project will be transferred to other current Optec projects (a recent decision by Optec management has been taken with this aim), in order to control in detail the quality of internal deliveries of documents and work products coming out from Optec departments or from external suppliers. More, Optec management intends to adopt the templates resulting from the experiment, in order to control any kind of requirements and work products, in any phase of the Optec life cycle (going from the customer contacts to the management of intermediate suppliers, and to the management of deliveries and acceptance tests);
- c) **the control of the entire development process.** Project planning, thanks to the accurate project plan structure required by the Commission in order to manage and monitor the progress of the PIE, has been another positive experience that Optec management intends to replicate in the new Optec projects, in order to control better than in the past the efforts and times spent. Optec management evaluates that the planning schema experimented in DOCPROVE is applicable, with minimal tailoring, in the other Optec projects;

and the **following product problems**:

- a) **project time and project cost** deviations. The reuse of the templates experimented in DOCPROVE will help Optec personnel in defining with more precision the structure of the work products also in terms of components, and therefore in estimating with lower risk of errors the efforts and times required for the development and testing. This improved capability should reduce the amplitude of deviations;
- b) **low reliability at delivery** time. Again the discipline of the above mentioned intermediate technical reviews, performed on each intermediate work product, will contribute to clean the work products from defects already in the phase when they are injected, reducing so the risks of defective final products delivered to the customer. The discipline experimented in DOCPROVE has been already transferred, with a good satisfaction of the customer, to another Optec

relevant project (from now on called ALPHA): in this case also the review and acceptance requirements have been preliminarily defined and agreed in co-operation with the customer representatives. This point is considered to be a remarkable spin-off of DOCPROVE project.

4.2. Business impact

The most relevant impacts on the business operation, on the basis of the achieved results, can be described through the following benefits:

- acceleration towards the Quality System's **ISO 9001 compliance certification**. The achieved results constitute already an improvement of Optec's Quality System, because the defined process models will become, after the project, a basic part of Optec's Quality System procedures (specifically: the project life cycle, the documentation templates; the review procedure, the data collection and metrics guides, the process assessment and improvement guide), characterised by a finalisation of the quality activities in support of customer needs and satisfaction.
- Better control of the planned **time to market**. Some models (Application Requirements, the clauses of requirement testability of System and Software Requirements, and the mentioned project planning schema), have been already applied in the mentioned ALPHA Optec project, passing also with success a joint review with the customer. This is a first (after DOCPROVE) evidence of an Optec's consolidated capability in eliciting the customer's requirements in a more complete and stable way, and in defining a detailed Work breakdown, which is the basis for development and testing plans which are more complete, more accurate, and therefore subject to lower estimation error, and more able to support a detailed control of the actual progress.
- The higher stability and completeness of requirements helps in achieving a **higher delivered quality**, thanks to less requirements changes in the final phases, and to quality control criteria defined in a finer detail (thanks to this aspect a satisfactory delivered quality has been already experienced in the ALPHA project).

4.3. Organisational impact

The improvements in project documentation are contributing to solve the following organisational problems:

- **Lack of infrastructure for project estimation**. The good level of completeness achieved in the definition of requirements allows to define plans in a deeper detail, based on a more detailed definition of Work breakdown structure, with positive impact on accuracy of estimates: a planning experience supported by detailed estimates has been the quality plan of ALPHA project.
- **Engineers tend to shortcut the standards due to overhead**. Often the standards are not so easy to apply in the daily activity, because of difficulties in interpretation. DOCPROVE has produced documentation guidelines (thanks to the use of ProDoc and ISO^{plus} tools [21], [22]) which contain, for each topics to be specified, a set of hidden help texts (with examples) which make the topics more easy to understand. Such DOCPROVE documentation guidelines have been produced independent of the type of system and software to be specified, thanks to the inspiration of IEEE standards. They are now available for other projects: a first usage outside DOCPROVE has been in the project ALPHA. Their ease of use, and the time saved because of the ready to use templates, sustain the motivation of engineers to apply them.
- **Low communication among projects**. The more accurate and updated documentation helps a better communication among projects, in the frequent cases where there are opportunities of specification, or design, or technical approach reuse: in the case of ALPHA project, the reuse did regard a plan schema already experienced in DOCPROVE.

4.4. Culture impact

«**Business driven quality culture**» has been the **slogan** inspiring the improvement project.

Properly planned training sessions were held **to transfer** to Optec people on one side a **business oriented quality view**, insisting on customer process driven elicitation of requirements, in order that the customer perceives the supplier as its business partner. In order to improve Optec abilities towards this objective, high care was put in teaching Optec people **how to help the customer in expressing his requirements and needs**.

With this aim, specific seminars have been organised where to transfer the know-how contained in the IEEE standard 1362-1998 [12] (which has been specifically prepared in support of elicitation of customer business rationales, and of customer needs), and to teach a model for process analysis and description. The effects of this tutorial action (as well as of the other

seminars held for other types of documents) has been controlled via specific questionnaires, where Optec people did express their satisfaction degree. An example of training seminar satisfaction results is provided in Figure 10.

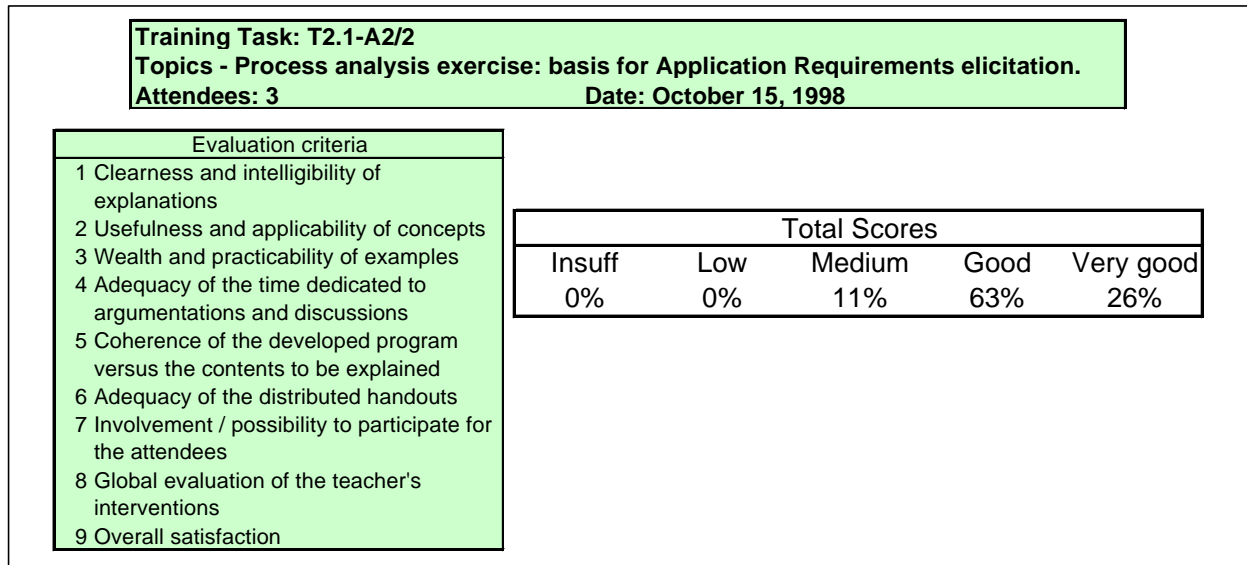


Figure 10 – DOCPROVE project: example of Employee Training Satisfaction Degree

On the other side **other facilitator factors** have been used during the experiment, in order **to facilitate a culture evolution and enrichment**: management commitment, two-ways communication and people involvement in setting-up the goals of the improvement project; use of customer feedback for planning future projects (this has been applied in the ALPHA project); internal dissemination of the improvement approach and results; provision of suitable tools and resources; tracking and managing the status of the project, and verification of improvements through planned measures.

4.5. Skills impact

The following skills and knowledge types have been generated by the improvement project (thanks to the training actions, and to the practical experimentation of new documentation practices in the context of a real project, the Baseline project):

- Customer process modelling, customer needs elicitation (experimented in DOCPROVE, and partially in project ALPHA).
- Ability to define and/or adapt system and software documentation templates (fully experimented in DOCPROVE, and in the Customer Needs, Quality Plan and Quality records forms of ALPHA project).
- Management of a systemic approach to projects (a specific care has been dedicated in DOCPROVE to the definition of System documents, under the persuasion that stable software requirements originate from well specified system requirements; this approach did allow to produce cultural and practical benefits also for system people in Optec).
- Matured awareness of advantages of good documentation on business results (this result has been achieved thanks to the mentioned training action regarding the analysis of customer processes and needs, and also thanks to the approach adopted for the definition of metrics, which were defined as indicators linked to defined business goals (see Figure 4, and document [17])). This approach consolidated the awareness that every project activity must be finalised to business goals.
- Good understanding of advantages of traceability and attributes of requirements as key ingredients of the approach, that help in performing good review and quality control activities (as a matter of fact, the degree of detail in quality control was facilitated in terms of correctness of review, and of time spent in review, by the use of traceability and requirements attribute). For example, the evaluation of completeness of requirements is practically impossible in medium/large projects, without the help of traceability.
- Knowledge of ISO Software process standards (such as ISO 12207 [1] and SPICE [7]) as ideal and clear guidelines for planning future incremental improvement actions (the knowledge accumulated regarding the two mentioned standards

is considered reusable for approaching the next process assessment and improvement activities in Optec, also at system level, in process areas different from the experimented documentation processes).

5. Key Lessons learned

The key lessons learned are considered to be the following ones.

5.1. Technical/technological point of view

From a software engineering point of view:

- The **process improvement** program is considered to have been well supported, by the **available international standards** such as ISO/IEC 12207 [1], ISO/IEC 15504 (SPICE) [7], for the activities experimented of **process definition and establishment**. Such standards may give to a novice reader the impression to be too much wide and complex. However, once their structure has been acquired (and this is easy) in such a way that the software engineering process specialist or the software engineer can find easily the section relevant to the process to be improved, such standards contain process model descriptions which are complete as well as sufficiently precise, concise, practical and reusable. We evaluate also that **process assessment** has been supported well by ISO/IEC 15504 ([7], parts 3, 4, 5) (see in section 4.1 of this paper the comments regarding the assessment).
However, as far as the document preparation activities are considered, the details available in ISO/IEC 15504 Part 5 [7], and in ISO/IEC 12207.1 [3] constitute very useful document skeletons, but they need to be completed by more rich guidelines, such as the ones provided by the selected IEEE standards ([12], [13], [14], [15], [16]).
- The **ami** methodology [5] is an application of **management good sense** to solve the problem of defining useful and **practical measurements of improvement**. The methodology provides good guidance for defining metrics which are linked to the selected business goals, especially with the suggested use of good sense questions which help in evaluating if a goal has been achieved.
- **Cheap but professional tools** are available, like ProDoc and ISO^{plus} ([21], [22]) which help significantly in providing a start basis in the **definition of the document templates and procedures** (their use generates a saving of effort and an increase of quality).
- **Traceability of requirements** and **specification of requirement attributes** are means that make easier and cheaper any activity of requirement quality control and change control. The level of details required for performing good controls is manageable only if a tool is used: the adopted **tool** [23] is evaluated to be **practical and easy to learn**.
- The **accuracy in documentation** is the best pre-requisite for **planning a project**, because well specified requirements support the production of more accurate effort and schedule estimates. This point has been already confirmed in Optec by the planning activity of project ALPHA, where specific care was put in defining the testability requirements, according to the guidelines derived from DOCPROVE experience.

5.2. Business point of view

A good awareness has matured in Optec about the following points:

- The initial usage (one month period) of the new Horus release developed by the Baseline project appears to be significantly **less defective** in comparison with past releases, even if, due to lack of past data, no quantitative comparison can be made. It is too early to say that this benefit (if confirmed by further use) is due to the care put in documenting the components impacted with the baseline project. We have to wait for a longer usage period, however it is interesting to record that already now the software team and system team say that system and software **specification quality is under better control**. But also at system level the Optec's systems documentation is evaluated to have received good improvements: being the system requirements evaluated to be much more solid and complete versus past experiences, and being therefore the deriving quality of mechanical and optical parts more controlled, Optec's people affirm that good documentation will produce **higher quality at delivery**.
- The development of an improvement project, like DOCPROVE, which was intentionally oriented to improve also the management of customer needs, produces results (as the document templates and document management procedures) which, thanks to the decision of incorporating them into the Optec's Quality System, makes the **Quality System more customer oriented** and **more ready to the ISO 9001 certification**.
- The achieved higher stability and completeness of requirements produces also positive effects on the ability to **keep the estimated delivery times and development costs**, for two main reasons: Optec's designer have experimented that good early definition of requirements keeps away the reasons of changes, and therefore the related additional cycles of

rework, and that good quality requirements help in preparing a development and qualification plan with much more details than in the past, and therefore less prone to errors in total estimates.

- Better technical specifications improve the **company image** towards the customers, as it was verified in a recent joint review with the customer of ALPHA project, showing a requirement specification document developed according to the DOCPROVE guidelines.

5.3. Strengths and weaknesses of the experiment

The following points should be remarked, as Optec's view of the usefulness of the experiment.

As **strength points**:

- the systematic **adoption of affirmed international standards** has allowed to define and put in practice a consistent approach (which includes improvement measurements), reusing consolidated experiences that other experts have put in the standards, and to avoid the risks of home-made approaches;
- use of **consolidated but simple, cheap tools** has allowed to concentrate the efforts on cultural, methodological and document content aspects rather than on support aspects;
- thanks to the fact that the approach has been experimented in Optec also at system level (and in part in the ALPHA project), and that the adopted process models (quoting [7] part 4, section. 5.2 5-th paragraph: "The model provided in ISO/IEC TR 15504-5 is a generic model that is designed to be applicable across all industry sectors and application domains"; quoting [7] part 4, section. 6.3: "A documented process supports repeatability of an assessment approach"), documentation templates and tools are well independent from the type of system and software to be developed, we can say that a **full replicability** of the experience is ensured first of all in other Optec's projects, but also in other companies, and other business domains. For example, the process analysis and customer needs analysis approach supported by the IEEE Std 1362-1998 [12] is really domain independent;
- various **facilitator factors**, already mentioned in section 4.4 (Culture impact).

As **weaknesses**:

- the **lack of documentation examples adherent to the specific company's business**, during the training sessions. If such examples were present, the effectiveness of the training and of the learning phases would have been even higher;
- the **lack of workgroup/workflow management tool and environment**: the use of such a kind of tool would have increased significantly the productivity of the team in the DOCPROVE project;
- difficulties in keeping the **schedules**, due in part to underestimation of the needed efforts, and mostly to variability of priorities due to urgency of other Optec projects.

It can be said that the **overall benefit** generated by the DOCPROVE project for Optec consists on the following points:

- to have imported/digested a **quality culture oriented** to elicit and to manage **customer needs**, to consolidate the related requirements and to formulate plans consistent with the requirements;
- the practical use of **schemas for expressing the requirements** starting from the contacts with the customer, and for using such requirements as reference (baseline) against which to make periodic checks of the status of a project;
- last but not least, the possibility to experiment in DOCPROVE (for the DOCPROVE project itself) a **project management schema**. We mean **the schema pushed by the European Commission**, which is based on: a well thought plan (contained in the project Annex), the successive formulation of a detailed plan, the Periodic Project Reports and Mid Term Report. The templates and guidelines, suggested by the Commission for such reports, are considered by Optec management a good tool which guides the intermediate reviews and controls, and which stimulates timely and criticising tracking thoughts and actions, during the way, on the important milestones to be achieved. We think that similar report schemas, properly adapted, will be very useful also for the other Optec's projects.

6. Conclusions and Future Actions

Optec will give serious consideration to the use of **SPICE** and of **ISO 12027** in support of future own improvement actions, and to ensure a further dissemination of that process culture to Optec people, as a **way to consolidate and sustain the management's and employees' motivation** to continue to improve the Optec's processes, and to accelerate the achievement of ISO 9001 certification.

In particular:

- To implement with priority, in each Optec future project, the preparation of the **Application Requirements** document, of the **System Requirements** document, and of the **Project Plan**, profiting of the DOCPROVE experience. Plans are seen as the key means to put the management in the position and attitude to allocate resources to next improvement actions.
- With the above aim, **derive** from the experimented templates, with priority to the above document types, **a set of refined templates** which can be proposed and discussed with the other Optec people, as work styles replicable in the other projects.
- Put the best care in transferring the results of the experiment into the Optec's Quality system (i.e. integrate the refined templates into the Quality System, involving the management and the designers in the necessary reviews and decisions)
- Apply, on future project, at least the measures of delivered defect indicators, `schedule_matching` and `spent_effort_matching`, with the related efficiency measures.
- Take into consideration the possibility to disseminate the experimented practices to our key suppliers and key customers, for making easier in the future the co-operations for managing the requirements and the suppliers' contracts.

References

- [1] 12207 Package - IEEE Standard for Industry Implementation of International Standard ISO/IEC 12207:1995, Standard for Information Technology - Software Life Cycle processes (the package includes the ISO/IEC 12207:1995 standard, and the Standards [3] and [4])
- [2] J-STD-016-1995 - EIA/IEEE Interim Standard for Information Technology, Software Life Cycle processes, Software Development, Acquirer-Supplier Agreement (Code: SH94377-NYF) (*this standard provides a set of guidelines for the documents to be produced during the life cycle*)
- [3] 12207.1-1997 - IEEE Standard for Industry Implementation of International Standard ISO/IEC 12207:1995, Standard for Information Technology - Software Life Cycle processes - Life Cycle Data
- [4] 12207.2-1997 - IEEE Standard for Industry Implementation of International Standard ISO/IEC 12207:1995, Standard for Information Technology - Software Life Cycle processes - Implementation Considerations
- [5] **ami** (Application of Metrics in Industry), Metric Users' Handbook - A quantitative approach to software management - The ami consortium c/o The **ami** User Group, CSSE, South Bank University, 103 Borough Road, London SE1 0AA, UK
- [6] Paulk M.C., Curtis B., Chrissis M.B., Weber C.V., 1995, The Capability Maturity Model, Guidelines for Improving the Software process, Carnegie Mellon University, Software Engineering Institute, SEI Series in Software Engineering, Addison Wesley
- [7] ISO/IEC TR 15504-n: Information Technology - Software Process Assessment (Parts 1 to 9)
- [8] 1028-1997 IEEE Standard for Software Reviews (code SH94592-NYF)
- [9] 1012-1998 IEEE Standard for Software Verification and Validation (code SH94625-NYF)
- [10] Basili V.R., Rombach H.D., 1988, «The TAME project: towards improvement-oriented software environment», IEEE Transaction on Software Engineering, vol.14, n. 6, June, pp. 322-331
- [11] ESSI PIE - Annex I - Project Programme - 27875 - DOCPROVE - Improvement of Software documentation practices, Version 1.2, 30-06-98
- [12] 1362 -1998 IEEE Guide for Information Technology--System Definition--Concept of Operations Document (code AD218-NYF)
- [13] 1233-1996 IEEE Guide for Developing System Requirements Specifications (Code: SS94407-NYF)
- [14] 830-1998 IEEE Recommended Practice for Software Requirements Specifications (Code: SH94654-NYF)
- [15] 1016-1998 IEEE Recommended Practice for Software Design Description (Code: SH11031-NYF)
- [16] 1063-1987 (Reaffirmed 1993) ANSI/IEEE Standard for Software User Documentation (Code: SS12039-NYF)
- [17] Moro P., Cicu A., 1999, «Improvement of software documentation practices: the approach adopted by the ESSI PIE (Process Improvement Experiment) DOCPROVE», Proceedings of Sixth European Conference on Software Quality (SOFTWARE QUALITY – THE WAY TO EXCELLENCE), April 12-16, 1999, Vienna (Austria) (see also the webs www.optec-srl.com, and www.metriqs.com)
- [18] 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology (Code: SH13748-NYF)
- [19] Moro P., Cicu A., 1999, «Improving software documentation: a customer oriented approach. The results of the ESSI PIE DOCPROVE project», Proceedings of the 3rd International Software Quality Week Europe (QWE'99), Brussels (Belgium), 1-5 November 1999 – Viewgraph presentation published as part of the Proceedings
- [20] ESSI PIE - Project number 27875 - DOCPROVE - The documentation process definition – Part 1 and Part 2 – Deliverable references: EDR1-1, EDR1-2, Version 1, 08-02-99
- [21] ProDoc V2.0 – User Manual (Handbook to Systems Development Documentation) – Set of documentation templates in support of software specifications and plans preparation – Software Productivity Centre, Vancouver, B.C., Canada
- [22] ISO^{plus} V1.0 – User Manual - Library of templates for the preparation of the Quality manual, the procedures and forms of an ISO 9001 compliant Quality System – Software Productivity Centre, Vancouver, B.C., Canada
- [23] RequisitePro, version 3.1 – User Manual - Rational Software Corporation

7. Acknowledgements

This work reflects experimentation performed in the framework of the ESSI PIE Project DOCPROVE, Contract No. 27875 [11], with the European Commission financial support.

Improving software documentation: a Customer oriented approach. The results of the ESSI PIE DOCPROVE project.

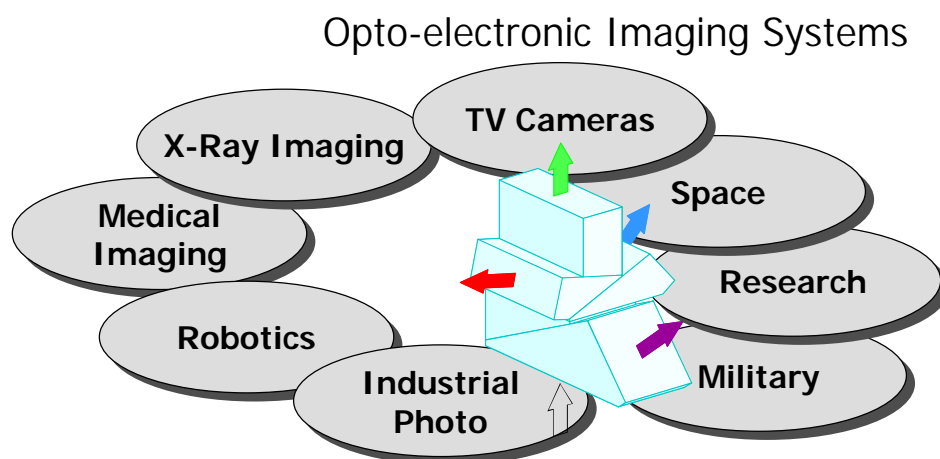
Pietro Moro, Optec srl
Antonio Cicu, MetriQs srl

3rd International Software Quality Week Europe
(QWE'99)
Brussels, 1-5 November 1999

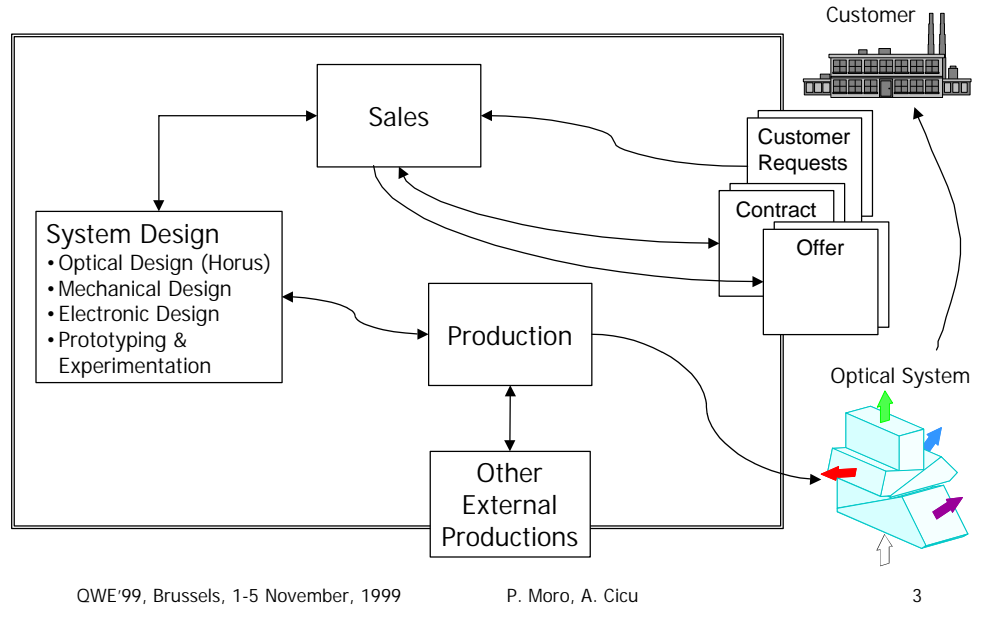
ESSI PIE Project 27875



Optec's Context



Optec's Business Cycle



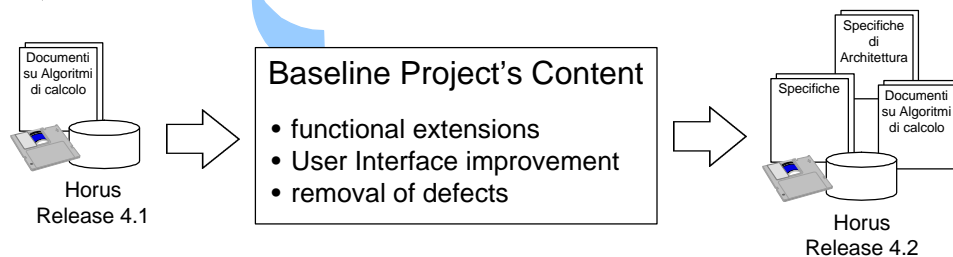
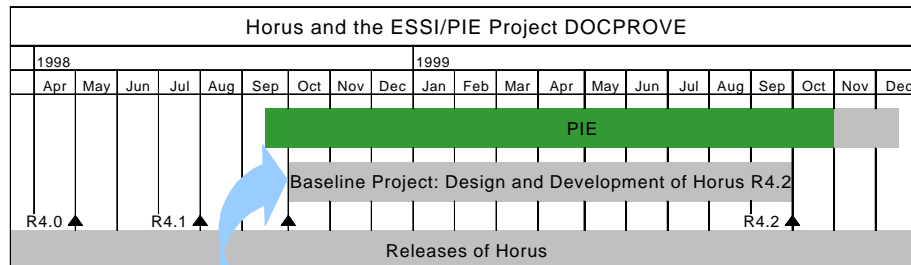
Horus: A Lens Design Program

- Optec's proprietary
- A strategic investment
- Optical design competence embedded
- Continuous development

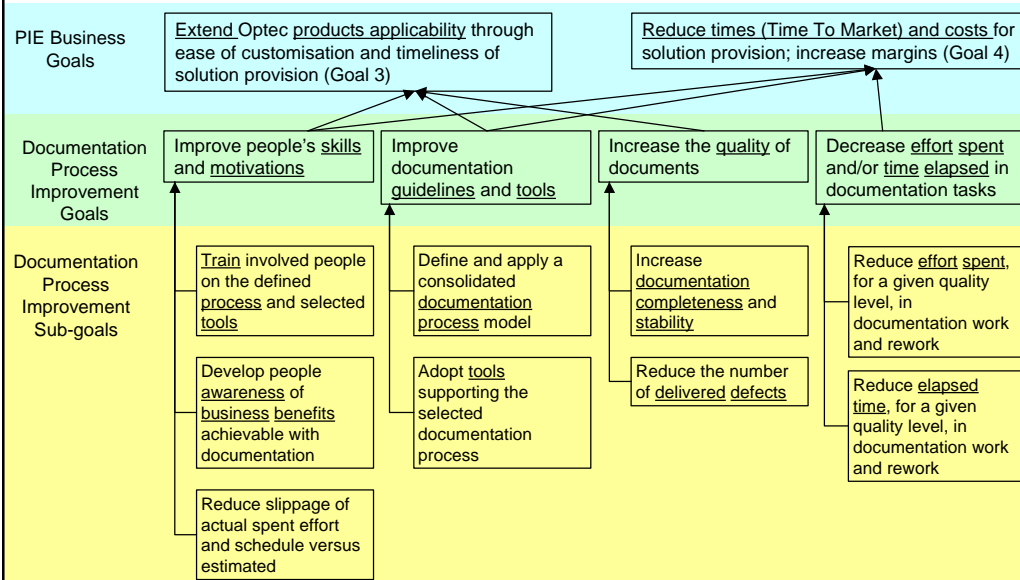


The objective of the PIE:
To improve software documentation process

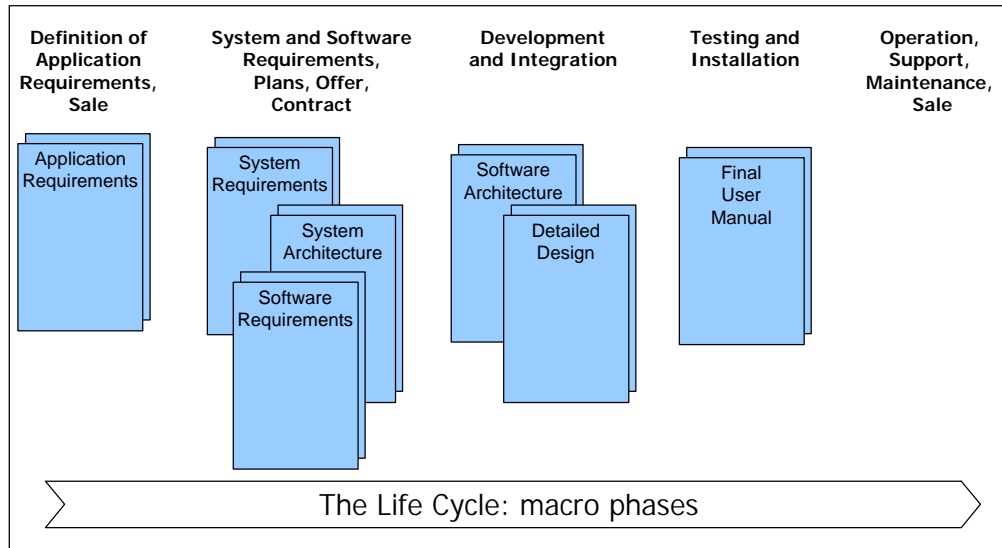
The Baseline Project



DOCPROVE PIE Goals Tree (*ami* method)



Experimented Documents

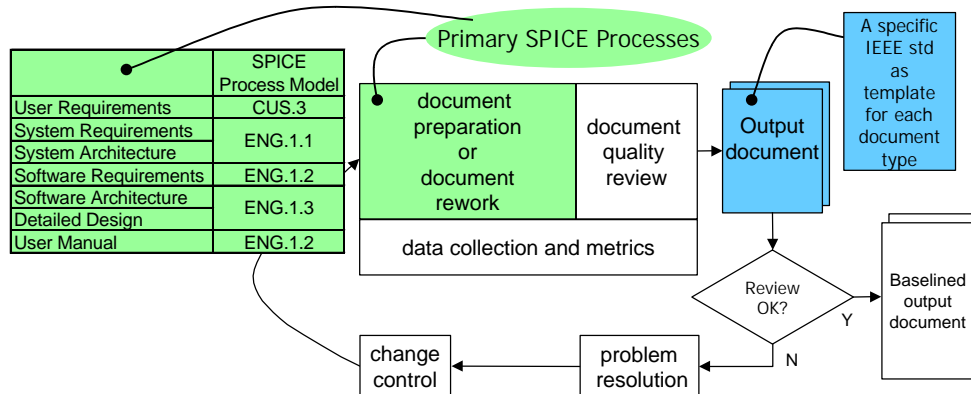


QWE'99, Brussels, 1-5 November, 1999

P. Moro, A. Cicu

7

The Process for Each Document (Primary Documentation Processes)

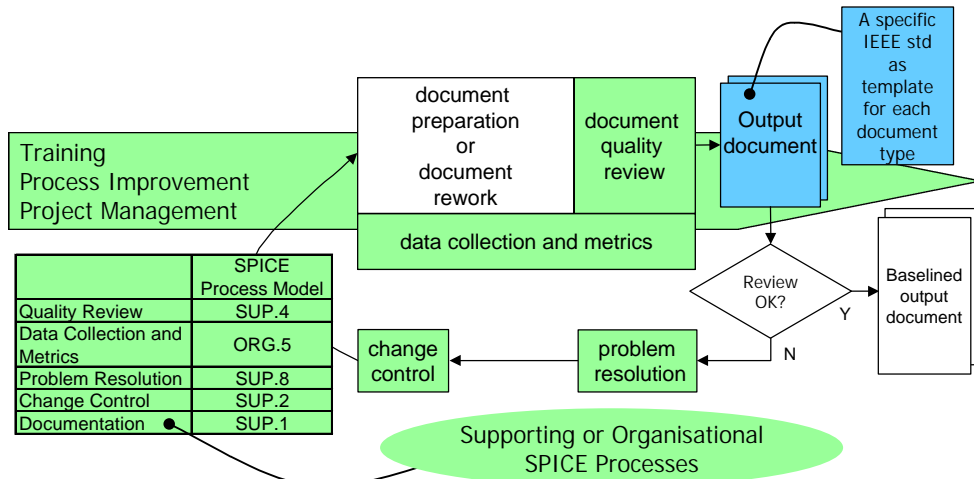


QWE'99, Brussels, 1-5 November, 1999

P. Moro, A. Cicu

8

The Process for Each Document (Supporting and Organisational Processes)

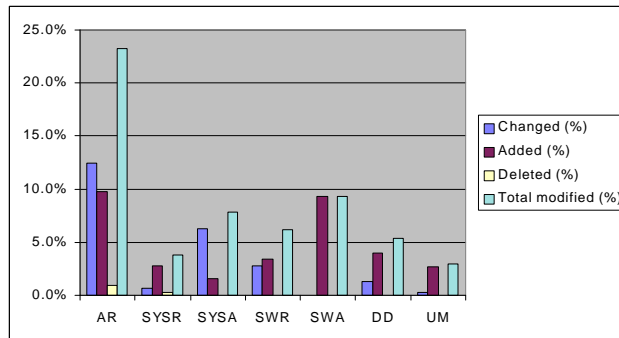


QWE'99, Brussels, 1-5 November, 1999

P. Moro, A. Cicu

9

Technical Impact: Documentation Completeness and Stability



| Experimented document | AR | SYSR | SYSA | SWR | SWA | DD | UM | Grand total w/out UM | w/UM |
|---------------------------------|-------|------|------|------|------|------|------|----------------------|------|
| Total requirements specified | 112 | 321 | 64 | 178 | 86 | 150 | 756 | 911 | 1667 |
| Requirements changed | 14 | 2 | 4 | 5 | 0 | 2 | 2 | 27 | 29 |
| Requirements added | 11 | 9 | 1 | 6 | 8 | 6 | 20 | 41 | 61 |
| Requirements deleted | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| Total requirements modified | 26 | 12 | 5 | 11 | 8 | 8 | 22 | 70 | 92 |
| Requirements changed (%) | 12.5% | 0.6% | 6.3% | 2.8% | 0.0% | 1.3% | 0.3% | 3.0% | 1.7% |
| Requirements added (%) | 9.8% | 2.8% | 1.6% | 3.4% | 9.3% | 4.0% | 2.6% | 4.5% | 3.7% |
| Requirements deleted (%) | 0.9% | 0.3% | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 0.2% | 0.1% |
| Total requirements modified (%) | 23.2% | 3.7% | 7.8% | 6.2% | 9.3% | 5.3% | 2.9% | 7.7% | 5.5% |

QWE'99, Brussels, 1-5 November, 1999

P. Moro, A. Cicu

10

Technical Impact: DOCPROVE Project SPICE Assessment

| Process | | Before the improvement project Process attributes | | | After the improvement project Process attributes | | |
|------------------------|--|--|----------------------------------|-----------------------------------|---|----------------------------------|-----------------------------------|
| SPICE process model ID | Process name | PA 1.1 Process performance | PA 2.1 Performance management | PA 2.2 Work product management | PA 1.1 Process performance | PA 2.1 Performance management | PA 2.2 Work product management |
| CUS.3 | Application Requirements document process | | | | | | |
| ENG.1.1 | System Requirements and System architecture document process | | | | | | |
| ENG.1.2 | Software Requirements document process | | | | | | |
| ENG.1.3 | Software Architecture and Detailed Design document process | | | | | | |
| ENG.1.2 | User Manual document process | | | | | | |
| SUP.1 | Documentation process | | | | | | |
| ORG.5 | Data collection and metrics (measurement) process | | | | | | |

LEGENDA

| | |
|------------------------|--|
| not achieved (N) | |
| partially achieved (P) | |
| largely achieved (L) | |
| fully achieved (F) | |

Business Impact

Relevant Benefits:

- Acceleration towards Quality System's ISO 9000 compliant Certification
- Time-to-Market
- Higher delivered quality

Organisational Impact

The project has contributed to solve the following problems:

- Lack of infrastructure for project estimation
- Engineers tend to shortcut the standards due to overhead
- Low communication among projects

Culture Impact

- Business driven quality culture, based on capability of understanding and analysing Customer's processes
- Increased capability of helping the Customer in expressing its needs
- Facilitators of culture evolution:
 - management commitment
 - people involvement in setting up the improvement goals
 - use of Customer feedback in support of project planning
 - internal dissemination

Culture Impact: Employee Training Satisfaction Degree: An Example

Training Task: T2.1-A2/2

Topics - Process analysis exercise: basis for Application Requirements elicitation.

Attendees: 3

Date: October 15, 1998

| Evaluation criteria |
|--|
| 1 Clearness and intelligibility of explanations |
| 2 Usefulness and applicability of concepts |
| 3 Wealth and practicability of examples |
| 4 Adequacy of the time dedicated to argumentations and discussions |
| 5 Coherence of the developed program versus the contents to be explained |
| 6 Adequacy of the distributed handouts |
| 7 Involvement / possibility to participate for the attendees |
| 8 Global evaluation of the teacher's interventions |
| 9 Overall satisfaction |

| Total Scores | | | | |
|--------------|-----|--------|------|-----------|
| Insuff | Low | Medium | Good | Very good |
| 0% | 0% | 11% | 63% | 26% |

Skills Impact

Skills and knowledge types generated by the project:

- Customer process modelling, Customer needs elicitation
- Ability to define and/or adapt system and software documentation templates
- Management of a systemic approach to projects
- Use of traceability and attributes of requirements
- Knowledge of ISO Software process standards (such as ISO 12207 and SPICE)

Lessons Learned

Technical / Technological Point of View

- **Process assessment and process improvement**
well supported by ISO/IEC 12207 and ISO/IEC 15504 (SPICE) stds
- **ami methodology**
management good sense applied for defining practical measurements
- **Cheap, but professional tools**
available for defining document templates and procedures
- **Requirements traceability and attributes**
support a more accurate quality control and change control
- **Accurate documentation**
prerequisite for planning a project
- **Experiment evaluated as replicable**
inside and outside Optec

Lessons Learned

Business Point of View

The experimented documentation produces:

- Higher quality at delivery
- Quality system more Customer oriented and more ready to ISO 9001 Certification
- Better ability to match estimated delivery times and development costs
- Better Company image

Optec srl

via Canova, 10
I-20017 RHO (MI)

tel: +39 0293501157

fax: +39 0293500207

moro.optec@agora.stm.it
<http://www.optec-srl.com>

How to test the EURO effectively



Andreas Rudolf / Rainer Pirker
AD Consultants
IBM Global Services Austria



Presentation overview

- Our experience
- Legal requirements
- EURO implementation (multistage plan)
- EURO ready versus EURO fit
- Technical phases
- Business transaction versus technical oriented testing
- Test activities per phase
- Unit tests
- Function tests
- System tests
- End to End tests (subsystem integration tests)
- Production supervision
- Problems seen
- Lessons learned

Our experience has been extended by large EURO and Y2K testing projects in different industries

Our Department is strongly focused on:

- Test Consulting (EURO, Y2K, e-business, new product developments etc.)
- Application Development Effectiveness Consulting (supporting the client to assess and improve their application development processes)

Our experience includes:

- Test consulting and test management for a dozen large EURO and Y2K projects
- Marketing and workshop activities for dozens of customer projects
- Test consulting in the finance, insurance, transport and utility industries

IBM Global Services Austria

The EURO phases and the timeframe



IBM Global Services Austria

The legal requirements: Council Regulation (EC) #1103/97 (The Article 235 Regulation)

Article 4

1. *Conversion rates shall be adopted as one euro expressed in terms of each of the national currencies of the participating Member States. They shall be adopted with six significant figures*
2. *The conversion rates shall not be rounded or truncated when making conversions*
3. *The conversion rates shall be used for conversions either way between the euro unit and the national currency units. Inverse rates derived from the conversion rate shall not be used*
4. *Monetary amounts to be converted from one national currency unit into another shall first be converted into a monetary amount expressed in the euro unit, which amount may be rounded to not less than three decimals and shall then be converted into the other national currency unit. No alternative method of calculation may be used unless it produces the same results.*

IBM Global Services Austria

The legal requirements: Council Regulation (EC) #1103/97 (The Article 235 Regulation) - cont.

Article 5

Monetary amounts to be paid or accounted for when rounding takes place after a conversion into the euro unit pursuant to Article 4 shall be rounded up or down to the nearest cent. Monetary amounts which are to be paid or accounted for which are converted into a national currency unit shall be rounded up or down to the nearest sub-unit or in the absence of a sub-unit to the nearest unit, or according to the national law or practice to a multiple or fraction of the sub-unit or unit of the national currency unit. If the application of the conversion rate gives a result which is exactly half-way, the sum shall be rounded up.

==> This regulation shall be binding in its entirety and directly applicable in all Member States (Luxembourg, 17 June 1997)

IBM Global Services Austria

The introduction of the EURO as legal tender for 11 countries is unique and cannot be handled like another currency

Article 235 regulations for conversion & rounding:

- ✓ Rates to six significant figures
- ✓ No Reciprocals
- ✓ Rounding to "the nearest number in the smallest sub-division of the local currency"
- ✓ Triangulation: Cross-conversion between National Currencies (NC) via Euro only
- ✓ No compulsion, No prohibition

EU regulations are only the start

→ **Base set of rules mandatory for Banking/Financial transactions only**

Organisations also need to cater for:

- Psychological pricing (e.g. 1.99) re-packaging or re-pricing?
- Literals (e.g. overdraft limits, trading authorisation)
- Accounting for rounding errors/differentials
- Rounding comparisons, e.g. difference between "Converted sum" v "sum of converted amounts"

Add to this

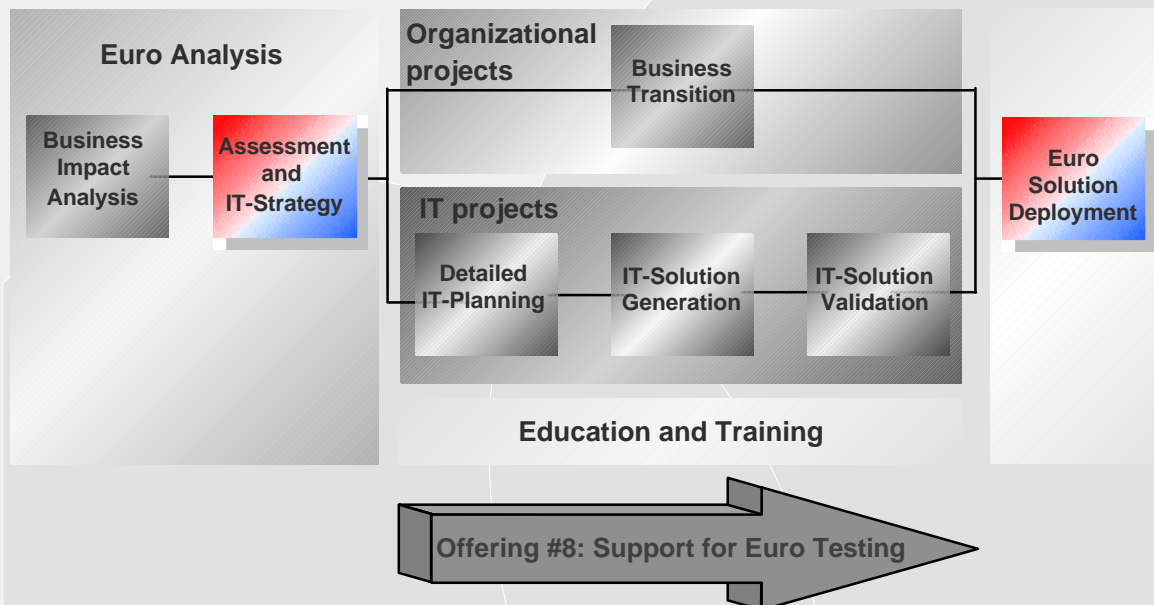
- *National Plans, Industry standards, Consumer pressure*
- *Accepted Practice vs Competitive Edge*

– **Not one but two changes:**

National Currency --> Dual-currency --> euro only

IBM Global Services Austria

IBMs recommended approach for the EURO is supported by eleven offerings



IBM Global Services Austria

Support for EURO Testing: The IBM Test offering implementing the EUROPath

Goal:

- detailed EURO test activities are performed to ensure the delivery of a completely and correctly migrated IT environment.

Key activities:

- Setting up a **EURO test plan** based on the client's existing test methods and processes
- Setting up a **technical test environment** and defining the test procedures
- Collecting all the **test cases and data**
- Detailed planning and performing of the **system integration / user acceptance tests**
- If necessary performing **stress and performance tests**
- **Evaluation** of all the results

Benefits:

- The Support for EURO Testing makes sure that the customer's IT management has done all the necessary tests to successfully complete his IT euro migration. Eventual errors and gaps are detected and can be solved or circumvented. The existing tests and test data are updated and can be used for further application maintenance work.

IBM Global Services Austria

Figures and planning from a sample insurance company to address the main objectives in EURO testing

Customer:

- large insurance holding in Austria

Goal:

- Planning and executing the changeover to EURO (inclusive managerial, technical and test planning)

Main figures:

- premium income 1.6b USD,
- employees 4.400,
- 9 regional-headquarters, 171 district-headquarters and branch offices
- all lines of business
- **subsidiaries and joint ventures in EU and non-EU-countries**

Duration:

- from the middle of 1998 until third quarter of 2002

IT-organisation:

- one IT-service organisation but different IT-systems
- no formal contract with other companies in the holding
- different releases of main IT-systems in the companies
- specific home grown solutions in the different companies

IBM Global Services Austria

Five main steps build the EURO implementation plan (multistage plan)

STEP 1:

- Requirements for **1.1.1999** are implemented (voluntary display of dual currencies and EURO fitness for some special applications)

STEP 2:

- Application supports **two decimals** and **currency indentifiers**

ATS 125,90-

STEP 3A:

- Requirements for **1.10.2001** are implemented (legal display of dual currencies - just in Austria)

ATS 123,90-

STEP 3B:

- Requirements for **1.1.2002** are implemented (EURO is base currency)

EUR 9,15-

STEP 4:

- Temporary solutions are **removed** (i.e. End of display of dual currencies)

EUR 9,15-

IBM Global Services Austria

This multistage plan can be subdivided in technical EURO requirements

EURO requirements for display of dual currencies:

- voluntary display of dual currencies for customer printouts (step 1)
- general company wide display of dual currencies (sometime in 1999)
- legal display of dual currencies (step 3A)
- cleanup of display of dual currencies (step 4)

EURO requirements for applications:

- a. conversion of programs and program logic
- b. interfaces conversion
- c. conversion and expansion of databases
- d. conversion of data

a) and b) are preconditions to reach step 2

a) to d) are preconditions to implement step 3B

IBM Global Services Austria

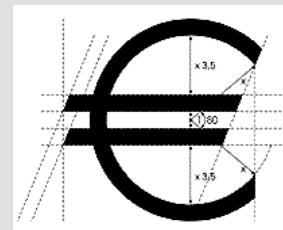
Only the state "EURO fit" for your application gives you the security that all EURO requirements are met

EURO readiness is granted for an application when the following requirements are fulfilled:

- passive requirements of EURO are implemented:
 - all amount fields have at least two decimal digits
 - all amounts or applications have a currency indicator, when necessary
- all special EURO requirements of the departments are taken into consideration and are implemented

An application has reached EURO fitness when the subject matter experts for this application certify it to be EURO fit:

- appropriate testcases have proven the EURO fitness from the business point of view
- an End to End test has been performed to check that all EURO requirements of this application are met



IBM Global Services Austria

The technical requirements can be seen as phases of the EURO changeover and for each phase milestones and activities for testing have to be planned



Technical requirements:

- Display of dual currencies
- Conversion of programs and program logic
- Interface conversion
- Database & data conversion

... planning for each phase:

Milestones

- Prerequisites for testing are available (HW, SW, documentation)
- Test goal achieved and/or sign off through subject matter experts

Activities

- Test preparation (documentation)
- Unit tests
- Function tests
- System tests
- End to End tests

(Each activity implies regression tests caused by errors and/or changes)

IBM Global Services Austria

For each application a test plan must be prepared to know which EURO activities have to be performed

Sample (matrix) plan of obligatory and optional activities for each EURO test phase:

| Phases / Activities | Display of dual currencies | Data & DB conversion | Program conversion | Interface conversion |
|--|----------------------------|---|--------------------|----------------------|
| Test preparation <i>(testcases, documentation)</i> | obligatory | obligatory | obligatory | obligatory |
| Unit tests | <i>optional</i> | <i>optional</i> | <i>optional</i> | <i>optional</i> |
| Function tests | obligatory | obligatory | obligatory | obligatory |
| System tests | <i>optional</i> | <i>optional</i> | <i>optional</i> | <i>optional</i> |
| End to End tests | not necessary | <i>tested with interface tests</i> | | obligatory |
| Supervision of production | obligatory | <i>ongoing checks in production environment</i> | | |

IBM Global Services Austria

The business transaction oriented test approach and the technical approach gives you the best of both worlds for testing

technical approach
(white & gray box testing)

- unit tests, system tests, EURO <-> national currencies calculation routines
- most errors came from newly written program adaptations
- prepared by technicians



business transaction oriented (black box)

- used for function tests, system tests, end to end tests, supervision of production
- allows you to focus on the most important business transactions
- prepared by subject matter experts

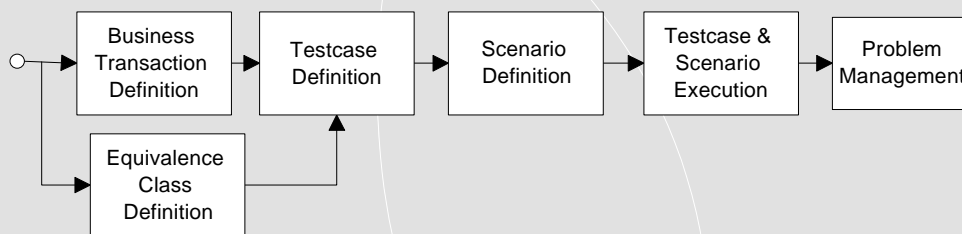
IBM Global Services Austria

Testcase administration and problem management databases were initially developed to support our Y2K test methodology

Our cornerstone tool suite is based on Lotus Notes and MS Access and integrates:

- Business transaction definition
- Test case definition
- Equivalence classes definition
- Test case and scenario execution
- Problem management

Methodology and Tool: From Test Case Definition through Execution



IBM Global Services Austria

Unit testing is recommended to identify problems early in the project

Goal:

- verify that a special module has met the (EURO) requirements

Test Method:

- white & gray box testing - code oriented

Focus of tests:

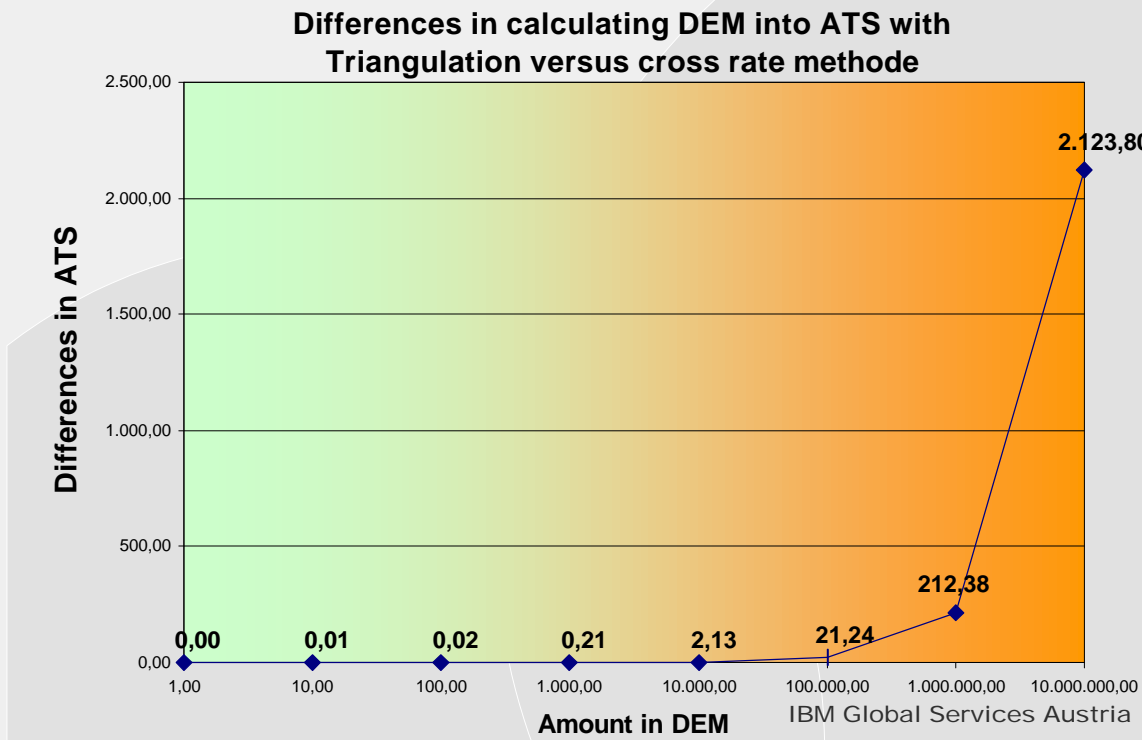
- central modules that are often applying amount calculations i.e. conversion modules
- database conversion routines

Special test areas:

- correct use of decimals and currency identifiers
- rounding problem
- triangulation versus cross or inverse rate

IBM Global Services Austria

If you do not use the proposed Triangulation method for converting currencies you always will get big differences



The purpose of function tests are to prove that the converted software produces logically equivalent results as before and after conversion of the EURO

Goal:

- verify the functional correctness and completeness of EURO converted programs

Test Method:

- black box testing - business transaction oriented



Focus of tests:

- all converted programs from the business transaction view

Special test areas:

- general functionality
- correct error messages
- correct navigating
- amounts appear clear and with decimals
- correct display of dual currencies
- correct print outputs
- optional: equivalence class method

The focus on EURO system tests is to check if the newly adapted programs and conversion routines can handle the required amount of data

Goal:

- verify that the required amount of data can be handled correct and within the required period of time

Test Method:

- gray & black box testing - code and/or business transaction oriented

Focus of tests:

- programs conversion routines that must handle big amounts of data
- programs and conversion routines that are time critical

Special test areas:

- performance tests
- volume tests



IBM Global Services Austria

End to End tests are the main overall effort in the changeover to EURO

Goal:

- verify that the whole system handles the EURO correctly. Ideally this should be done in form of a series of complete tests of the interfaces of all applications - in practice this may not be possible

Test Method:

- black box testing - business transaction oriented

Focus of tests:

- all important and critical business transactions covering the whole system

Special test areas:

- time warp tests will be necessary to simulate the introduction of the different EURO steps
- side effect: lots of synergies to the Y2K tests and test environment
- complex test scenarios must be generated
- special test environment needed

Year 2000

IBM Global Services Austria

There are a lot of synergy effects between EURO and Y2K tests

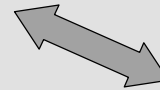
Fundamental differences:

- Year 2000: no changes to functionality (more technical focus)
- EURO: business impact, new functionality to support mandatory requirements

Synergy effects:

- inventory and test environment
- identification of critical processes/ applications, prioritisation
- reuse of business transactions
- identification of cross-business processes and applications
- test methods and test data
- regressions test tool kit and scripts
- documentation and communication
- project management methodology

Y2K



Problem areas:

- avoid converting and testing together - testing 2 "unknowns"
- rigorous change control and configuration management
- do not reuse test cases - they are special for Y2K and EURO

IBM Global Services Austria

Supervision of production is necessary to constantly proof the correct use of EURO in the beginning of the changeover period

Goal:

- verify that in production the EURO is handled correctly, since not all tests are possible to be performed in the test environment, test effort versus value of test does not match and not all tests can be done because of ressource and time restriction

Test Method:

- spot check - business transaction oriented

Focus of tests:

- important and critical business transactions
- not previous tested interfaces
- print outs

Special test areas:

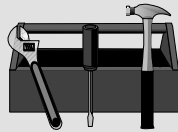
- production data different from test data
- main calculations
- customer printouts

IBM Global Services Austria

Tools are used to improve testing productivity and to support data manipulation

Capture/replay tool

- QAHiperStation™ from CompuWare (mainframe applications)
- QA Partner™ from Segue (client/server systems)



Advantage of using a capture/replay tool for regression tests

| Activity | Time using QA Partner™ | Time without QA Partner™ |
|------------------------------|------------------------|--------------------------|
| Test case creation | 160 h | 150 h |
| Correcting generated scripts | 5 h | |
| Initial learning effort | 5 h | |
| Test case execution once | 10 h | 40 h |
| Total | 180 h | 190 h |
| Test execution 3 more time | 30 h | 120 h |
| Total | 210 h | 310 h |

Data manipulation

- File-AID™ from CompuWare (prepare and verify test data)
- File-AID/Data Ager™ from CompuWare (data aging)

IBM Global Services Austria

Problems seen in the beginning of the project

- Business Management decisions are necessary for every EURO phase but not often seen
- Customer had no defined business transactions and no test cases
- Internal and external interfaces got often overseen in the beginning
- Dedicated test environment occupied by Y2K tests
- New product development projects cause a lot of additional problems i.e. the EURO project also has to define test procedures for them
- Very limited resources for testing
- Almost no education for the testers
- Message of the customer in the beginning: "EURO is only conversion"
- No reviews of the created testcases with the subject matter experts
- Testcases covered often only standard situations

IBM Global Services Austria

Lessons learned

What went right?

- Using test cases based on business transactions (Black box test strategy)
- White/Gray box testing for currency calculation routines
- Establish a test database for all business transactions and testcases used in EURO tests
- Dedicated test team
- Test planning from the beginning of the project

What went wrong?

- Effort to integrate new product development projects into the EURO project was underestimated
- Reviews MUST be done in every phase of testing - regardless communication is a problem
- A test database, test environment and test tools must be installed at the early beginning of the test
- Overall test effort was underestimated by customer management

IBM Global Services Austria

Testing: What is unusual about a Euro project?

- Pervasive change - company wide
- Most systems affected
- From minimal change through to competitive leverage
- Conversion and cutover
- Fall back
- Severely constrained timescales
- External and internal influences and dependencies
- Year 2000
- The combination is unique ...

IBM Global Services Austria

Open questions?

Ask them right now!

or if they come up later contact us:



- Andreas Rudolf
- e-mail: Andreas_Rudolf@at.ibm.com
- Phone: +43-1-1706-4347
- Fax: +43-1-1706-2393
- Web: www.geocities.com/Vienna/Strasse/7559/ IBM Global Services Austria



- Rainer Pirker
- e-mail: Rainer_Pirker@at.ibm.com
- Phone: +43-1-1706-4163
- Fax: +43-1-1706-2393

Quality Week Europe '99

How to Apply for Funding from the IST Programme

-

Otto Vinter

Software Engineering Mentor

Tel/Fax: +45 4399 2662, Mobile: +45 4045 0771

vinter@inet.uni2.dk <http://inet.uni2.dk/~vinter>

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Information Society Technologies (IST)

Creating a user-friendly information society

- 3600 MEUR over 4 years
- 7 Action lines and Supporting Measures
 - Research actions (RTD)
 - Take up measures (best practice, trials)
 - SME special measures
- Two calls for proposals per year
- 50-100% funding.
- Projects 1-3 years
- Partners from more than one country

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Information Society Technologies (IST)

Actions lines

- KA I Systems and services for the citizen
- KA II New methods of work and electronic commerce
- KA III Multimedia content and tools
- KA IV Essential technologies and infrastructure
- CPA V Cross-programme themes
- FET VI Future and emerging technologies
- RN VII Research networking
- ISM VIII IST support measures

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Information Society Technologies (IST)

Information at: <http://www.cordis.lu/ist>

- Call text
- Work programme
- Guide for proposers
- Evaluation manual
- Guidelines for evaluators

Current call - October 1st

- Specific subset of actions called for
 - check: <http://www.cordis.lu/ist>
- Deadline for proposals January 17th
- Next call expected January 15th

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Information Society Technologies (IST)

Who can participate ?

with Community funding

- EU member states
- Candidate member states
- Iceland, Liechtenstein, Norway, Israel, Switzerland

without Community funding

- any country on a project by project basis, if in conformity with the interests of the Community

At least 2 partners

Less than 50% of budget in one country

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Information Society Technologies (IST)

Selection process

1. Call for proposals
2. Proposal submission
3. Administrative check on eligibility
4. Evaluation by external experts
5. Summary reports by panel of experts
6. Priority list of proposals suitable for funding
7. Financial/administrative check and negotiations with participants
8. Selection decision
9. Contract signature

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Information Society Technologies (IST)

Selection criteria

1. Scientific/Technological quality and innovation

- quality of research, degree of innovation, adequacy of approach

2. Community added value and contribution to EU policies

- European dimension of the problem, added value of the consortium, contribution to EU policies

3. Contribution to Community social objectives

- improving the quality of life and health and safety, employment, preserving/enhancing the environment

4. Economic development and S&T prospects

- usefulness and range of applications, quality of exploitation plans, strategic impact, dissemination strategies

5. Resources, partnership, management

- quality of management/project approach, quality of partnership, appropriateness of the resources

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Information Society Technologies (IST)

Type of action

examples

Selection criteria

(weight threshold)

| | S&T quality innov. (w t) | Com. added value (w t) | Contrib. to Com. soc.obj. (w t) | Econom. Develop. S&T pro. (w t) | Resources Partnersh. & Mgmt. (w t) |
|------------------|-----------------------------------|---------------------------------|--|--|---|
| RTD (one-step) | 4 3 | 1 2 | 1 - | 2 3 | 2 2 |
| FET (Pro-active) | 4 3 | 1 1 | 1 - | 2 - | 2 2 |
| Take-up | 4 - | 1 2 | 1 - | 2 - | 2 4 |

Scores: 0 - Unsatisfactory 1 - Poor 2 - Fair
3 - Good 4 - Very good 5 - Excellent

© 1999-09-20

Otto Vinter
Software Engineering Mentor



What Is a Good IST Proposal ?

- Relevant subject within the scope of the call
- Clear and realistic goals
- Measurable results
- Realistic and efficient work plan
- Realistic budget and costing
- Elements in selection criteria covered
- The right partners
 - researchers as well as end-users
 - good distribution over countries
 - well-defined roles and responsibilities
 - some you already know or trust
- An IST project must be just as well prepared and argued as any other application for funding in your company

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Contract Negotiation Phase

- Based on proposal ratings and evaluator comments
- Changes in the proposal must be introduced
- Be prepared for cuts in the proposed budget
- The amended proposal text becomes part of the contract (Annex 1, Project Programme)
- Otherwise a standard CEC contract (no changes)
- Establish partner agreements (contracts)
- Project start date app. July 1st
- 1st advance payment 60 days after signature by CEC

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Executing an IST Project

- **Maintain effective project control**
 - resource consumption
 - progress
 - results
- **Maintain effective communication**
 - technical level
 - management
 - externally (dissemination)
- **Follow the work plan**
 - be prepared for changes (flexible)
- **Handle conflicts internally**
- **When changes are necessary**
 - submit a revised plan for approval
 - extensions of duration are possible
 - the budget cannot be increased

© 1999-09-20

Otto Vinter
Software Engineering Mentor



Final Important Advice

Participation should be a company decision


- **Commission funding does not mean: no commitment**
- **Internal demand for the results**
- **Performed as any other project**
 - allocation and prioritisation of resources
- **Internal “champions”**
 - dedication
 - ability to motivate and present
 - avoid political games / power struggles

© 1999-09-20

Otto Vinter
Software Engineering Mentor




Slide 1



Performance 2000: Ensuring Consistent Website Performance

Scott Yara
Vice President of Marketing
Sandpiper Networks

Slide 2





Agenda

- Today's Internet
 - The Problem
 - High Expectations
 - Today's Infrastructure
- Emerging Standards for Improved Performance
 - The Highway
 - The First Mile
 - The Last Mile
 - Content Delivery Networks
 - Additional solutions

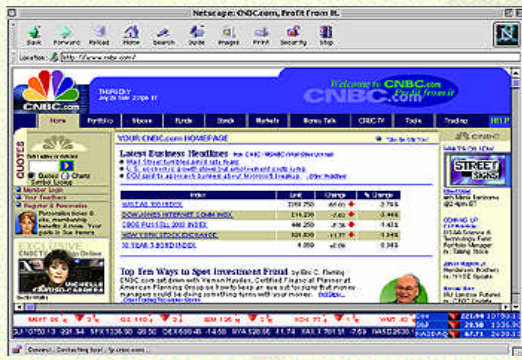
Slide 3

The Problem - Web Page Downloading

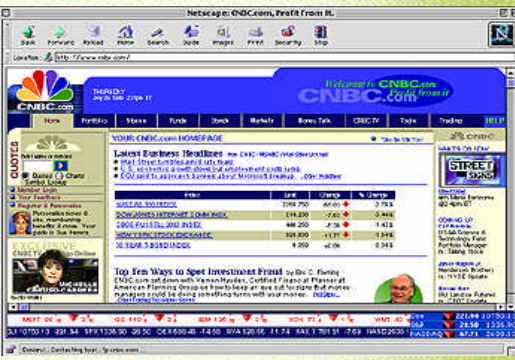




E-commerce dollars lost due to bad performance.




Without Footprint



With Footprint


Slide 4

High Expectations



- Internet traffic is doubling every 100 days
- Voice, data, video, audio all coming down same pipe -- all billed separately
- Full color, full motion, full screen video
- CD quality audio
- Real-time simulations, 3D virtual reality
- Virtual libraries and data archives
- Geographically dispersed groups to share specialized equipment

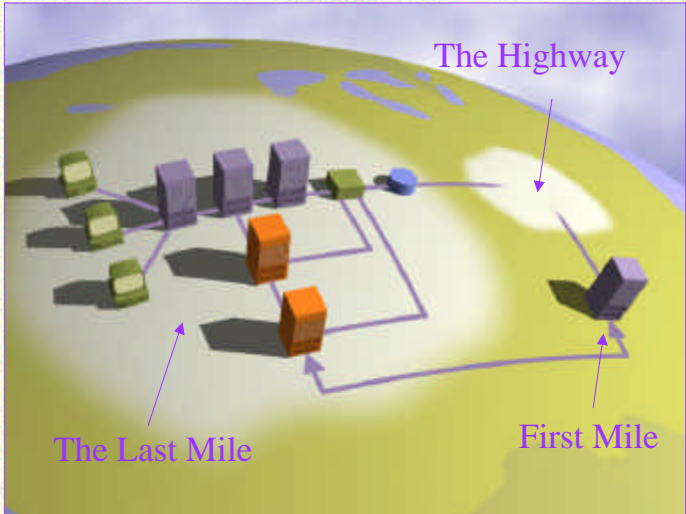

Today's Infrastructure



- No single entity controls the Internet
- 1 million largely private networks
- 45 million computer hosts
- "Best effort" - no performance guarantees
- Dynamic and unpredictable network conditions
- E-commerce transactions are aborted due to long wait-times:
 - \$4.35B loss in US, 1998*


* Zona Research

The Internet of Tomorrow: A Closer Look



The diagram illustrates network infrastructure on a globe. A central cluster of purple and orange blocks represents a network core. A purple line, labeled 'The Highway', connects this core to a single purple block on the right. A purple line, labeled 'The Last Mile', connects the core to a cluster of green and orange blocks on the left. A purple line, labeled 'First Mile', connects the core to a single purple block on the right. The globe is shown in a 3D perspective with a blue sky and green landmasses.

Slide 7



Emerging Highway Landscape

- Fiber Optics (DWDM) - Lucent, Ciena
- ATM / SONET - Fore, Cisco
- IP Packet Gateways - Lucent, Cisco
- Gigabit / Terabit Routers - Extreme, Juniper
- Network Caches - Inktomi, CacheFlow
- Satellite Providers - Hughes, PanAm Sat
- Peering Centers - AboveNet, Equinix

Slide 8



Emerging First Mile Landscape

- Industrial-Strength Data Centers - Exodus
- Premium Transit Connections - InterNap
- High-end Servers - Sun
- Load-balancing Switches - Aleton
- Storage Appliances - Network Appliance
- Distributed Databases - Oracle
- Content Delivery Networks - Sandpiper

Emerging Last Mile Landscape




- Faster Analog Modems - Ascend
- xDSL - Rhythms
- Cable Modems - @Home
- Satellite Networks - Teledesic
- Access Caching - Inktomi

Three routes to broadband

| | | |
|---|---|---|
| CABLE Cable modem networks operate like giant local-area networks (LANs) — the service is always on | TELEPHONE Digital Subscriber Line (DSL) offers throughput up to 1Mbps and uses existing phone lines | WIRELESS Though they are unproven, wireless networks could be fastest to deploy — they don't require digging cables |
|---|---|---|


SOURCE: Carol Wilson, Inter@Cive Week

Content Delivery Networks



CDN's improve performance by serving content closer to end-users

- Hundreds of servers
- Multiple networks
- Patent-pending technology
- Intelligent probes
- Professionally managed

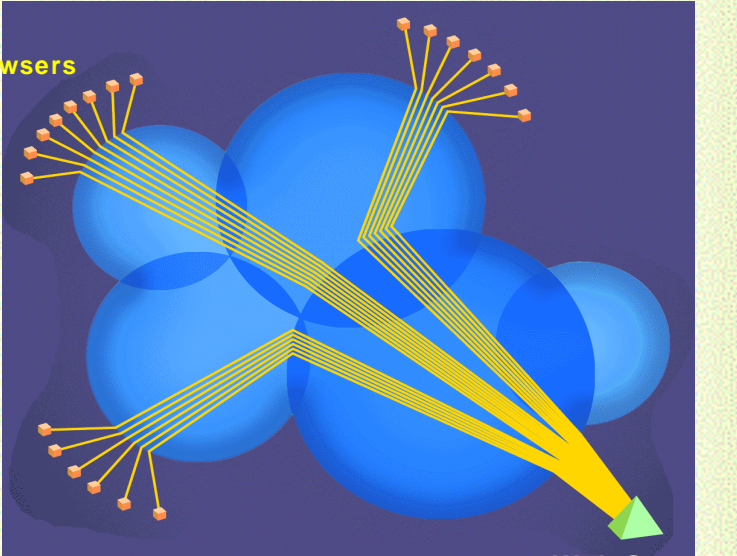



Content Delivery Networks




- Outsourcing solution for publishers
 - massive, shared infrastructure
 - reserved resources
 - serves all clients
 - adapts to traffic conditions
- Additional publisher services
 - audio/video support
 - authentication services
 - dynamic content support

A Popular Web Site

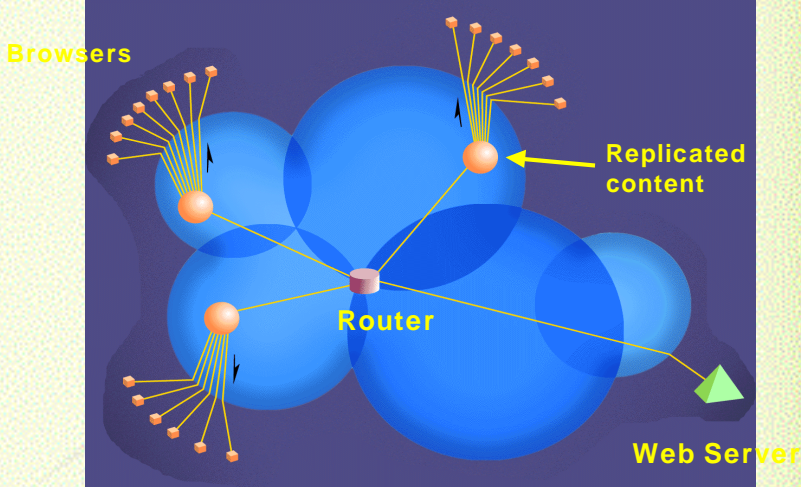


The diagram illustrates a popular web site's architecture. It features a central cluster of blue spheres representing web servers. A large green arrow points from the servers towards the right, labeled "Web Servers". On the left, multiple orange spheres represent browsers, with yellow lines connecting them to the server cluster. The label "Browsers" is positioned near the top left of the browser cluster.

CDNs - Replicating Content Closer to Users




- Reduces load on server
- Avoids network congestion



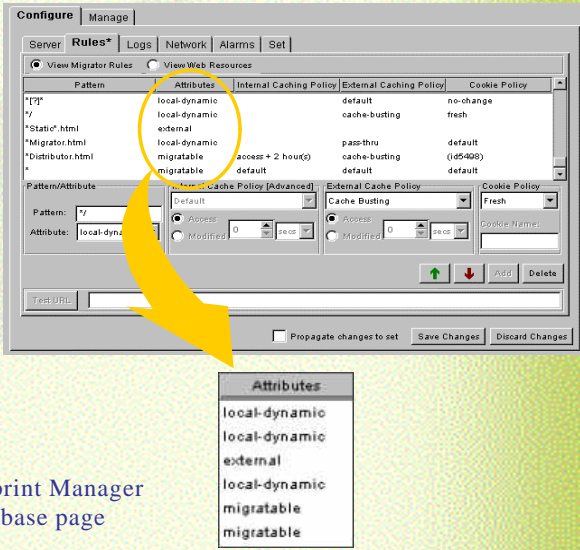
The diagram illustrates a Content Delivery Network (CDN) architecture. It features a central 'Router' (red circle) connected to multiple 'Replicated content' nodes (orange circles). These nodes are further connected to 'Browsers' (represented by groups of small orange circles) and a 'Web Server' (green triangle). The replicated content nodes are distributed geographically to be closer to the users (browsers).

How Footprint Works



1. Content Preparation

- Publishers identify which resources should be served by the Footprint CDN
 - Publisher uses Footprint Preparation tool to modify HTML (static rewriting)
 - Tool uses rule base to determine which resources to serve
- Process is extremely simple



The screenshot shows the 'Footprint Manager rules base page' with a table of rules. A yellow arrow points from the 'Attributes' column of the table to a dropdown menu showing the available attributes.


| Pattern | Attributes | Internal Caching Policy | External Caching Policy | Cookie Policy |
|-------------------|---------------|-------------------------|-------------------------|---------------|
| *[?] | local-dynamic | default | no-change | |
| *[?] | local-dynamic | cache-busting | fresh | |
| *Static*.html | external | | | |
| *Migrator.html | local-dynamic | access + 2 hour(s) | pass-thru | default |
| *Distributor.html | migratable | cache-busting | default | (id5408) |
| * | migratable | default | default | default |

Attributes dropdown menu:

- local-dynamic
- local-dynamic
- external
- local-dynamic
- migratable
- migratable

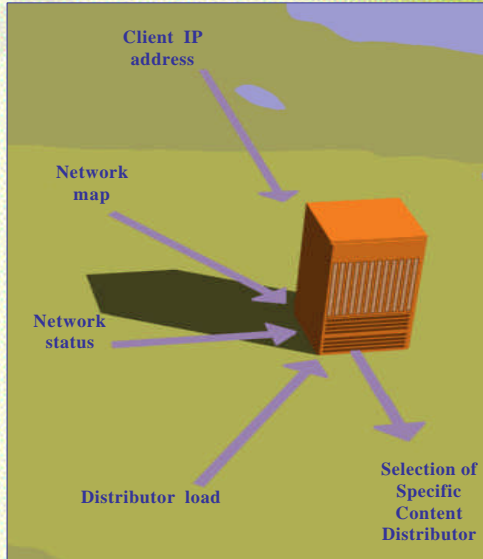
Footprint Manager rules base page

How Footprint Works




2. Client Rendezvous

- Sandpiper uses a process called Best Distributor Selection to direct end-users to the optimal location
 - Intelligent probes constantly monitor the state of each network
 - Content Distributor (CD) selection based upon CD loading, Client network location and network status
 - ▲ The Best Distributor is selected dynamically by a modified DNS server

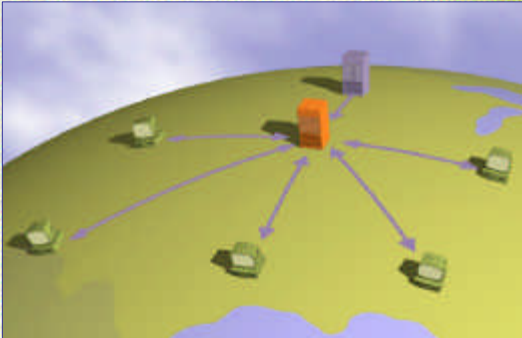


How It Works





3. Content Delivery

- Once an end-user is directed to the fastest location, the content is delivered from the Content Distributor
 - Content Distributor checks if currently cached content is fresh and if not retrieves fresh content from origin server



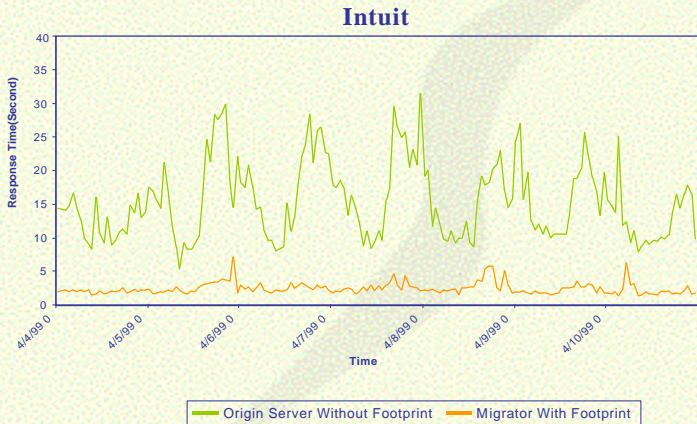

Case Study: Intuit



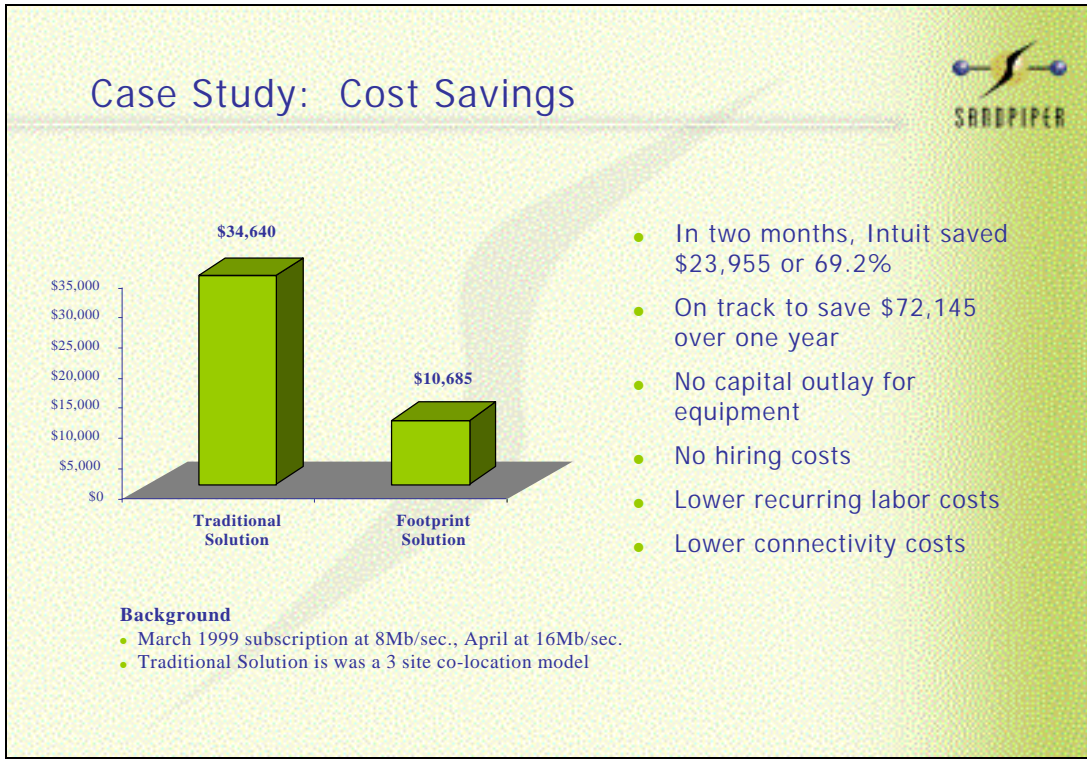
- Intuit experiences significant increases in web site traffic during tax season
- The company used Footprint to handle peak-traffic demands for distributing products such as Quicken, Turbo Tax, and QuickBooks Pro
- Footprint solution integrated seamlessly into Intuit's existing infrastructure
- Web site performance improved while costs decreased

“Footprint helps us guarantee that our customers get the upgrades fast and consistently regardless of heavy traffic requests.” - Rick Parkinson, Intuit


Performance Improvements




- 50.8% average improvement in response time during peak hours
- 40.3% overall average improvement in response time



All Content Types




Good performance requires quality delivery for each of the multiple content types that can make up a web page:



The screenshot shows a web browser window with several content elements highlighted by yellow circles and labeled with arrows:

- Rotating Banners:** A banner at the top of the page.
- Targeted Ads:** A small advertisement on the right side of the page.
- Personalized Settings:** A small box on the right side of the page.
- Premium Content:** A large article or video player in the main content area.
- Real-time Information:** A section at the bottom of the page.
- Streaming Media:** A video player on the left side of the page.
- Search Queries:** A search bar at the top left of the page.

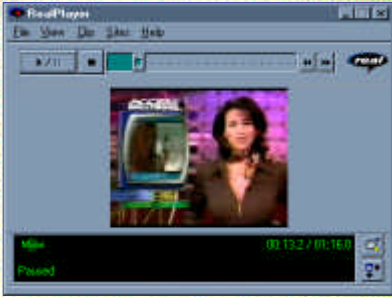
Streaming Content



- Audio and video files are extremely bandwidth intensive
- Today's architecture causes degradation of streaming files


Streaming with Footprint

- Scales to vast numbers of viewers without changes to servers
- Ensures more consistent and better quality of stream
- Requires no investment in hardware or software
- Content provider does not need to own Real licenses



The screenshot shows a RealPlayer window with a video player. The video shows a woman speaking. The player has a progress bar at the bottom showing 00:132 / 01:160.

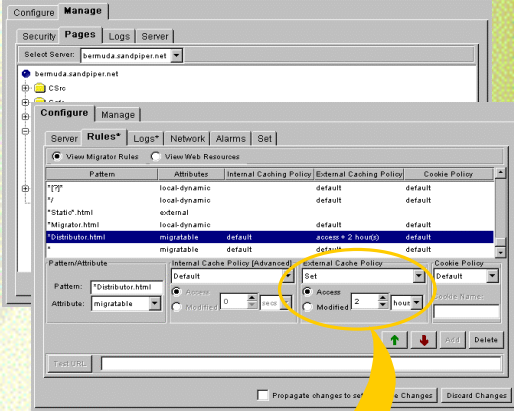
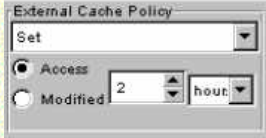
Publishing Tools



- Sandpiper provides several easy-to-use tools that ensure content served by the Footprint Network is always up-to-date:
 - GUI
 - Script
 - Auto-publish


Benefits

- Control freshness on CDN
- Control freshness in ISP deployed caches

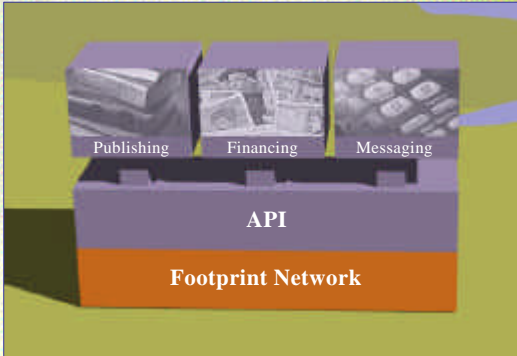
Auto-publish tool

Open Architecture



Sandpiper's open architecture gives Content Providers ultimate flexibility in leveraging the Footprint network to build:

- Custom applications
- Industry-specific solutions
- Value-added service offerings




Content Delivery Solution Providers

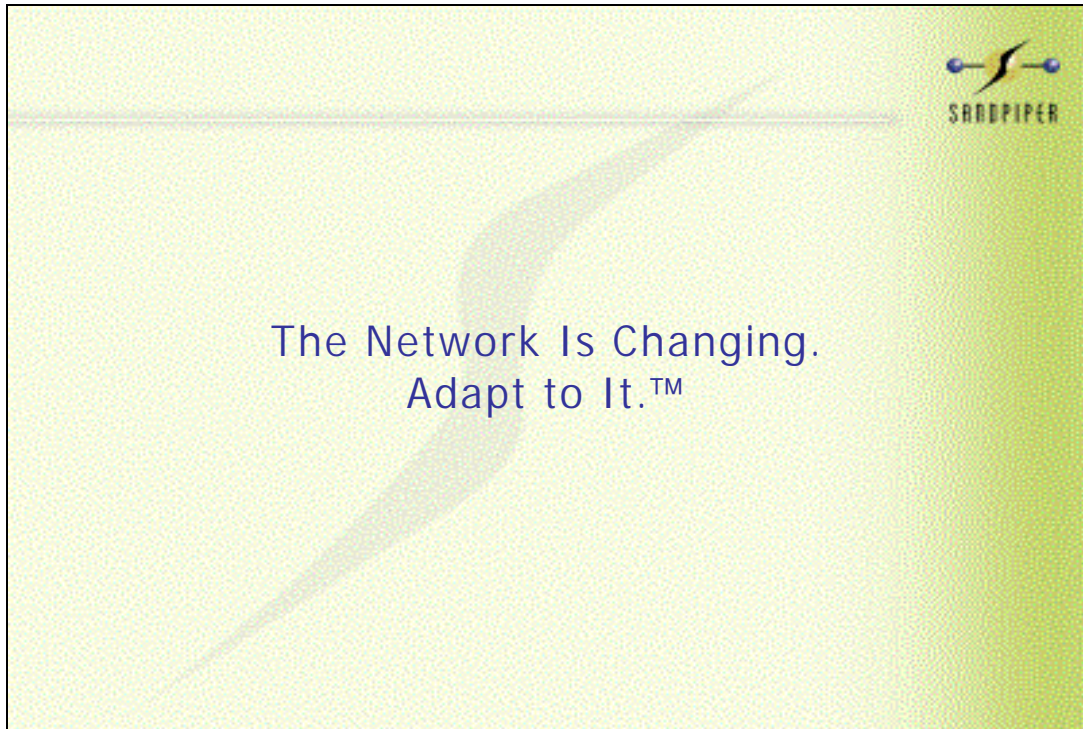


- IBeam
- Skycache
- Intervu
- Digital Island
- Mirror Image
- Exodus

Highlights



- Quality of service is becoming more difficult to achieve
- CDNs improve the performance and reliability of web content delivery by distributing content closer to end users
- A host of solutions are available to improve performance
- As the Internet grows, content delivery solutions will become mission critical for all web sites





Performance 2000: Ensuring Consistent Website Performance

Scott Yara
Vice President of Marketing
Sandpiper Networks

Agenda



- Today's Internet
 - The Problem
 - High Expectations
 - Today's Infrastructure
- Emerging Standards for Improved Performance
 - The Highway
 - The First Mile
 - The Last Mile
 - Content Delivery Networks
 - Additional solutions

The Problem - Web Page Downloading



\$4,370,000,000

E-commerce dollars lost due to bad performance.



Without Footprint



With Footprint

High Expectations



- Internet traffic is doubling every 100 days
- Voice, data, video, audio all coming down same pipe -- all billed separately
- Full color, full motion, full screen video
- CD quality audio
- Real-time simulations, 3D virtual reality
- Virtual libraries and data archives
- Geographically dispersed groups to share specialized equipment

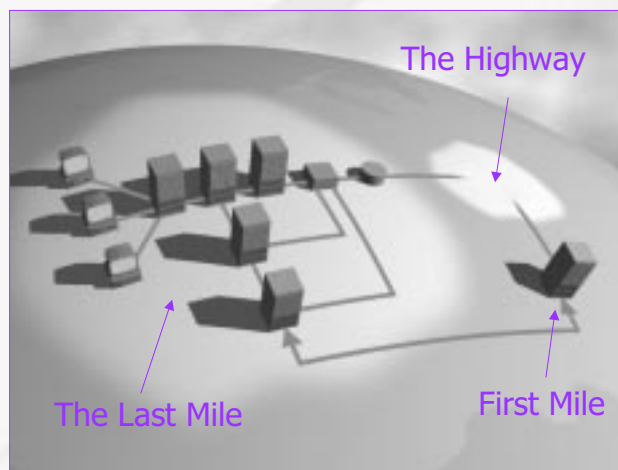
Today's Infrastructure



- No single entity controls the Internet
- 1 million largely private networks
- 45 million computer hosts
- "Best effort" - no performance guarantees
- Dynamic and unpredictable network conditions
- E-commerce transactions are aborted due to long wait-times:
 - \$4.35B loss in US, 1998*

* Zona Research

The Internet of Tomorrow: A Closer Look



Emerging Highway Landscape



- Fiber Optics (DWDM) - Lucent, Ciena
- ATM / SONET - Fore, Cisco
- IP Packet Gateways - Lucent, Cisco
- Gigabit / Terabit Routers - Extreme, Juniper
- Network Caches - Inktomi, CacheFlow
- Satellite Providers - Hughes, PanAm Sat
- Peering Centers - AboveNet, Equinix

Emerging First Mile Landscape

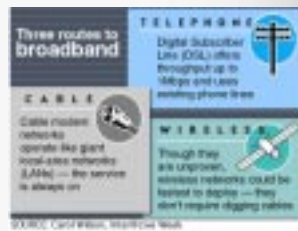


- Industrial-Strength Data Centers - Exodus
- Premium Transit Connections - InterNap
- High-end Servers - Sun
- Load-balancing Switches - Aleton
- Storage Appliances - Network Appliance
- Distributed Databases - Oracle
- Content Delivery Networks - Sandpiper

Emerging Last Mile Landscape



- Faster Analog Modems - Ascend
- xDSL - Rhythms
- Cable Modems - @Home
- Satellite Networks - Teledesic
- Access Caching - Inktomi



Content Delivery Networks



CDN's improve performance by serving content closer to end-users

- Hundreds of servers
- Multiple networks
- Patent-pending technology
- Intelligent probes
- Professionally managed

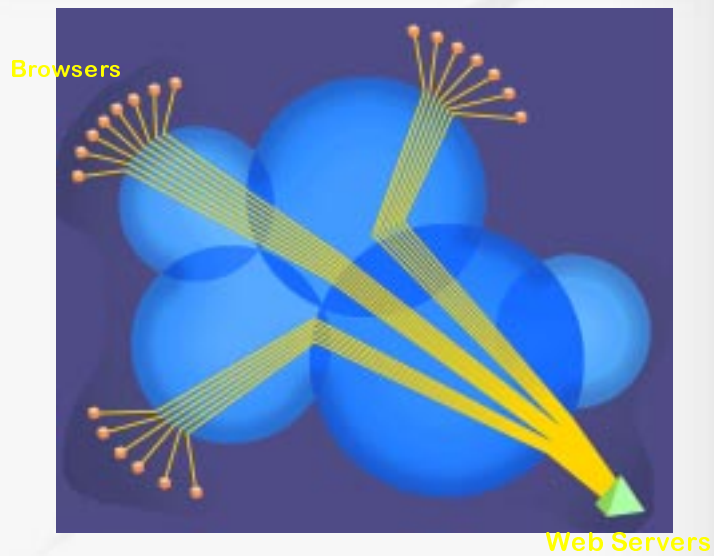


Content Delivery Networks



- Outsourcing solution for publishers
 - massive, shared infrastructure
 - reserved resources
 - serves all clients
 - adapts to traffic conditions
- Additional publisher services
 - audio/video support
 - authentication services
 - dynamic content support

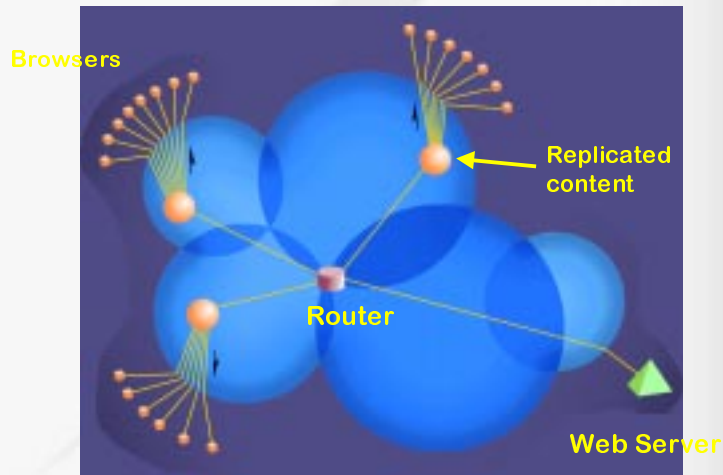
A Popular Web Site



CDNs - Replicating Content Closer to Users



- Reduces load on server
- Avoids network congestion

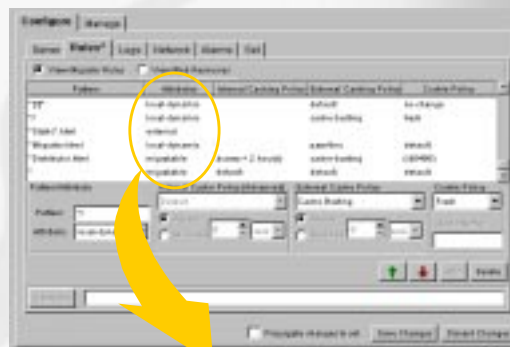


How Footprint Works



1. Content Preparation

- Publishers identify which resources should be served by the Footprint CDN
 - Publisher uses Footprint Preparation tool to modify HTML (static rewriting)
 - Tool uses rule base to determine which resources to serve
- Process is extremely simple



Footprint Manager rules base page

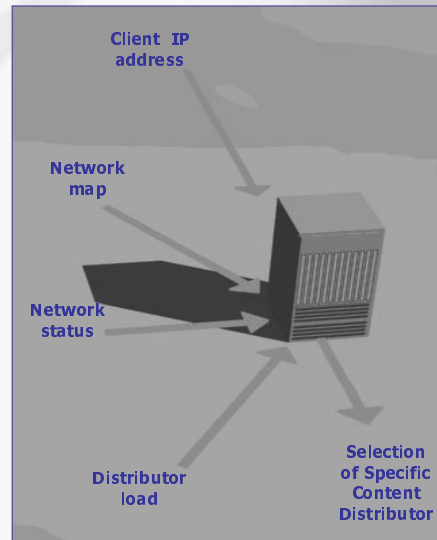
Attributes
local-dynamic
local-dynamic
external
local-dynamic
inmutable
inmutable

How Footprint Works



2. Client Rendezvous

- Sandpiper uses a process called Best Distributor Selection to direct end-users to the optimal location
 - Intelligent probes constantly monitor the state of each network
 - Content Distributor (CD) selection based upon CD loading, Client network location and network status
 - ▲ The Best Distributor is selected dynamically by a modified DNS server



How It Works



3. Content Delivery

- Once an end-user is directed to the fastest location, the content is delivered from the Content Distributor
 - Content Distributor checks if currently cached content is fresh and if not retrieves fresh content from origin server



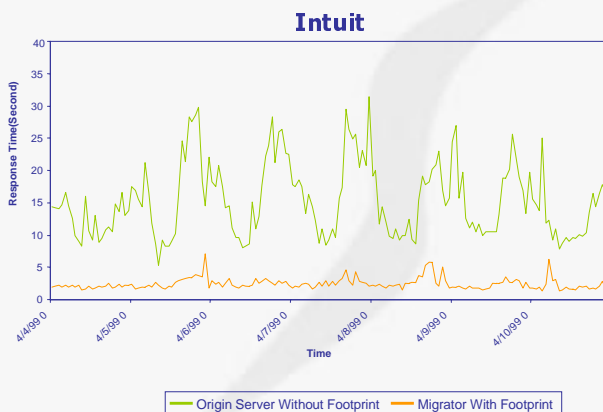
Case Study: Intuit



- Intuit experiences significant increases in web site traffic during tax season
- The company used Footprint to handle peak-traffic demands for distributing products such as Quicken, Turbo Tax, and QuickBooks Pro
- Footprint solution integrated seamlessly into Intuit's existing infrastructure
- Web site performance improved while costs decreased

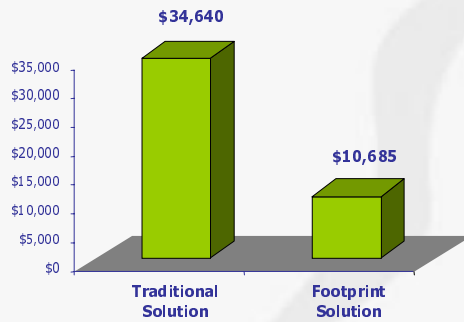
"Footprint helps us guarantee that our customers get the upgrades fast and consistently regardless of heavy traffic requests." - Rick Parkinson, Intuit

Performance Improvements



- 50.8% average improvement in response time during peak hours
- 40.3% overall average improvement in response time

Case Study: Cost Savings



- In two months, Intuit saved \$23,955 or 69.2%
- On track to save \$72,145 over one year
- No capital outlay for equipment
- No hiring costs
- Lower recurring labor costs
- Lower connectivity costs

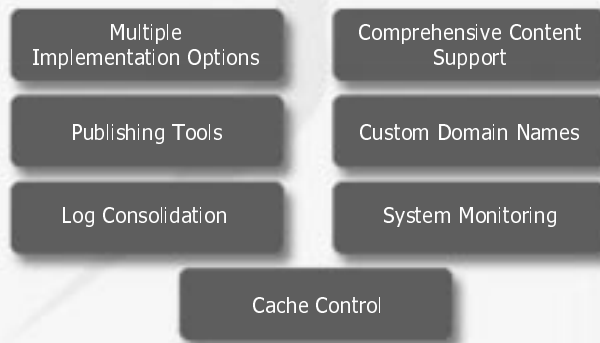
Background

- March 1999 subscription at 8Mb/sec., April at 16Mb/sec.
- Traditional Solution is was a 3 site co-location model

Comprehensive Solution



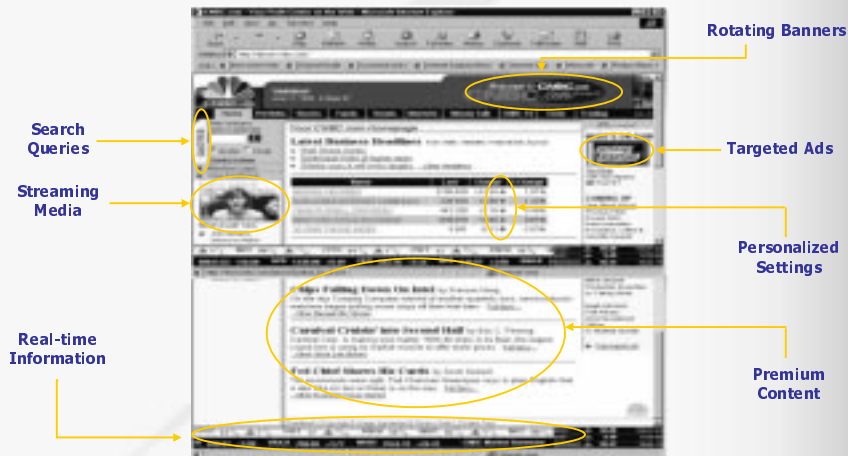
Footprint is a complete solution - making it easy to implement, manage, and track your content throughout the network.



All Content Types



Good performance requires quality delivery for each of the multiple content types that can make up a web page:



Streaming Content



- Audio and video files are extremely bandwidth intensive
- Today's architecture causes degradation of streaming files

Streaming with Footprint

- Scales to vast numbers of viewers without changes to servers
- Ensures more consistent and better quality of stream
- Requires no investment in hardware or software
- Content provider does not need to own Real licenses



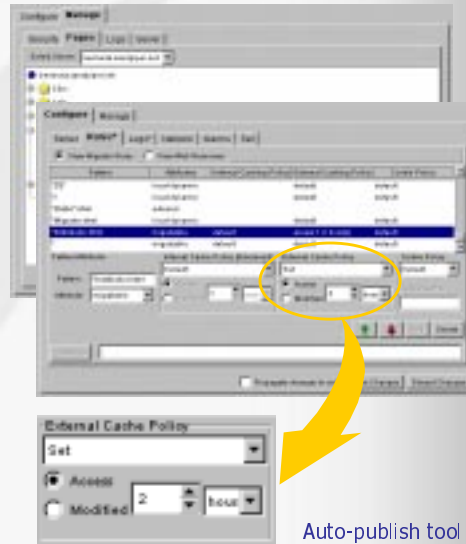
Publishing Tools



- Sandpiper provides several easy-to-use tools that ensure content served by the Footprint Network is always up-to-date:
 - GUI
 - Script
 - Auto-publish

Benefits

- Control freshness on CDN
- Control freshness in ISP deployed caches



Open Architecture



Sandpiper's open architecture gives Content Providers ultimate flexibility in leveraging the Footprint network to build:

- Custom applications
- Industry-specific solutions
- Value-added service offerings



Content Delivery Solution Providers



- IBeam
- Skycache
- Intervu
- Digital Island
- Mirror Image
- Exodus

Highlights



- Quality of service is becoming more difficult to achieve
- CDNs improve the performance and reliability of web content delivery by distributing content closer to end users
- A host of solutions are available to improve performance
- As the Internet grows, content delivery solutions will become mission critical for all web sites



The Network Is Changing.
Adapt to It.™

E-Commerce Reliability and Web Site Testing

Software Research, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA
evalid@soft.com



eValid -- The Internet Quality Authority



WebSite Quality Factors

- Time / Change
- Structure
- Content
- Accuracy and Consistency
- Response Time and Latency
- Performance



eValid -- The Internet Quality Authority



WebSite Architectural Aspects

- HTML
- Java, JavaScript
- Cgi-Bin (Perl, etc.)
- Database Access
- Multi-Media



eValid -- The Internet Quality Authority



Assuring WebSite Quality Automatically

- Browser Independent
- No Buffering, Caching
- Fonts and Preferences
- Object Mode
- Tables and Forms
- Frames



eValid -- The Internet Quality Authority



CAPBAK/Web

Test Enabled Web Browser

- Script Capture/Replay
- Script Creation/Replay
- Site Validation and Verification
- Performance Timing
- Content Validation



eValid -- The Internet Quality Authority



CAPBAK/Web CHARACTERISTICS

- Fully Featured Web Browser
- Built on IE 4.n, 5.n
- Full User Capture/Playback Engine
- "C" Script Language
- Script Recording, Script Editing
- Complete Support For:
 - ActiveX, Frames, XML, DHTML, SSL
 - FORMs, Modal Dialogs
 - JavaScript, VBScript, etc.
- Multiple Playback Capability



eValid -- The Internet Quality Authority



CAPBAK/Web USER FEATURES

- **Intuitive GUI Via PullDowns**
- **Test Wizards**
 - Link Test
 - Form Test
 - Button Test
- **Validation Functions**
 - Text [Fragments]
 - Images [Fragments]
- **Timing Functions**
 - User Controlled Clock
 - User Defined Alarms
- **Reporting**
 - Timer Log
 - Message Log
 - Event Log



eValid – The Internet Quality Authority



CAPBAK/Web Browser



eValid – The Internet Quality Authority



CAPBAK/Web Help Pulldown



eValid – The Internet Quality Authority



CAPBAK/Web Pulldown, View



eValid – The Internet Quality Authority



CAPBAK/Web Pulldown, Preferences



eValid – The Internet Quality Authority



CAPBAK/Web Pulldown, Timer, Set Alarm



eValid – The Internet Quality Authority



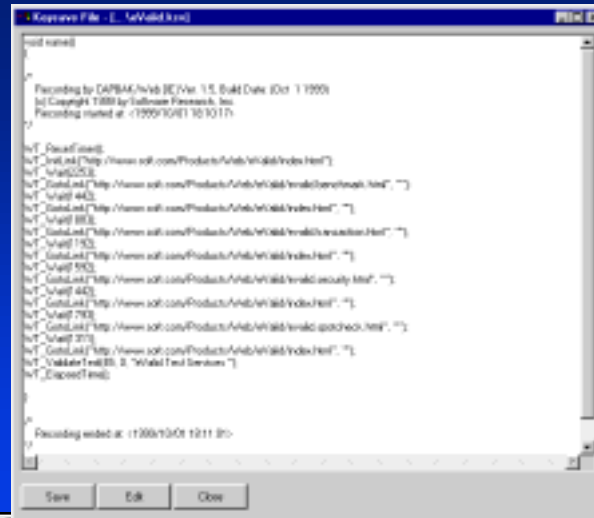
CAPBAK/Web Pulldown, Wizards



eValid – The Internet Quality Authority



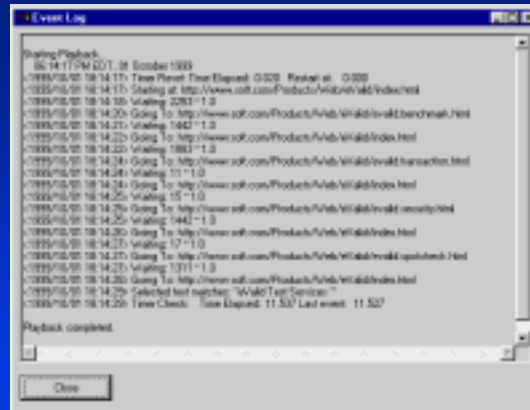
CAPBAK/Web Sample Keysave File



eValid – The Internet Quality Authority



CAPBAK/Web Sample Event Log File



```
Event Log
Starting Playback
12/14/11 10:11:00 AM EDT: 01: Escobar 1999
1/1999/12/01 18:14:17: New Planet Time Elapsed: 0:00: Remaining: 0:00
1/1999/12/01 18:14:17: Starting at http://www.sdr.com/Products/Valid/eValid/index.html
1/1999/12/01 18:14:18: Waiting: 2761 ~ 1.0
1/1999/12/01 18:14:20: Going to http://www.sdr.com/Products/Valid/eValid/benchmark.html
1/1999/12/01 18:14:21: Waiting: 1482 ~ 1.0
1/1999/12/01 18:14:22: Going to http://www.sdr.com/Products/Valid/eValid/index.html
1/1999/12/01 18:14:22: Waiting: 1883 ~ 1.0
1/1999/12/01 18:14:24: Going to http://www.sdr.com/Products/Valid/eValid/transaction.html
1/1999/12/01 18:14:25: Waiting: 131 ~ 1.0
1/1999/12/01 18:14:26: Going to http://www.sdr.com/Products/Valid/eValid/index.html
1/1999/12/01 18:14:25: Waiting: 15 ~ 1.0
1/1999/12/01 18:14:26: Going to http://www.sdr.com/Products/Valid/eValid/secure.html
1/1999/12/01 18:14:25: Waiting: 1441 ~ 1.0
1/1999/12/01 18:14:26: Going to http://www.sdr.com/Products/Valid/eValid/index.html
1/1999/12/01 18:14:25: Waiting: 17 ~ 1.0
1/1999/12/01 18:14:27: Going to http://www.sdr.com/Products/Valid/eValid/quickbooks.html
1/1999/12/01 18:14:27: Waiting: 1311 ~ 1.0
1/1999/12/01 18:14:28: Going to http://www.sdr.com/Products/Valid/eValid/index.html
1/1999/12/01 18:14:28: Selected test together: "Valid Test Services"
1/1999/12/01 18:14:28: Test Check: Time Elapsed: 11:52: Last event: 11:52
Playback completed
```

eValid -- The Internet Quality Authority



CAPBAK/Web Interface to SMARTS

- **Hierarchical Test Tree**
 - 1000's of Tests
 - Test Groups Nested Without Limit
- **Multiple Test Execution Modes**
 - Selected Test [Sub-]Tree
 - Variable Outcomes
- **PASS/FALL Log Data**
 - Direct PASS/FALL Statistics
 - Export Data to Spreadsheet
 - Export Data to DataBase

eValid -- The Internet Quality Authority





eValid -- The Internet Quality Authority

**Subscription Test TeleServices
Custom Web Site Testing and Validation
Web Site Quality Consulting**



eValid -- The Internet Quality Authority



WEB MARKETPLACE SIZE

- **44.1 Million On-Line Shoppers in 1999**
- **128 Million On-Line Shoppers in 2002**
- **E-Commerce ~\$327 Billion in 2002**
- **1,500,000 Domain Names Registered**
- **\$1B/year in site analysis, performance and management tools (Dataquest)**



eValid -- The Internet Quality Authority



eValid MARKET TARGETS

- **Web Editors/Creators**
 - Main CAPBAK/Web Buyer
 - Single Short Term eValid Purchase
- **Web Masters**
 - Likely CAPBAK/Web Buyer
 - Several Middle Term eValid Purchases
 - Some Long-Term eValid Purchases
- **Web Site Managers**
 - Possible CAPBAK/Web Buyer
 - Multiple Long-Term eValid Purchases
 - TestSuite Development Support
 - Consulting Services
- **IT Directors**
 - Possible CAPBAK/Web Buyer
 - Multiple Long-Term eValid Purchases
 - TestSuite Development
 - Web Quality Consulting Services



eValid -- The Internet Quality Authority



WEB SITE PERFORMANCE / QUALITY ANALYSIS PERSPECTIVES

- **On-ISP**
 - Runs Local to WebSite
 - Static/Local Analysis
 - Possible Outbound Link Checks
- **ISP-to-ISP**
 - Runs Remote from Website
 - Dynamic Analysis
 - Possible Quality Analysis
- **Client-to-ISP (eValid's View)**
 - Runs on Client Through Web to Site
 - 100% Users' Perspective
 - 56.6 Kbps Dialup Modem Standard
 - Multiple TimeZones
 - Client Creation/Verification of Tests



eValid -- The Internet Quality Authority



eValid SERVICE OPTONS -- Frequency

- **24 x 7 x 365**
- **Timing Interval**
 - Semi-Hourly (15 Minutes) 2,880 Tests/Month
 - Hourly 720 Tests/Month
 - Four-Hourly 180 Tests/Month
- **TimeZone Coverage**
 - USA
 - Eastern
 - Central
 - Mountain
 - Pacific
 - Non-USA
 - Pacific Rim
 - Europe



eValid -- The Internet Quality Authority



eValid SERVICE OPTONS -- Custom Testing

- **Consulting on WebSite Testing**
- **TestSuite Development**
 - Onsite
 - Remote
- **TestSuite Execution**
 - 24 x 7 x 365 "standard" Service
 - Customer Schedule
- **"Tiger Team" WebSite Assurance**



eValid -- The Internet Quality Authority



eValid SERVICE OPTONS -- WebSite Validation

- **BenchMark Test Service**
 - How Fast Does a Sequence of URLs Download?
 - 1-6 or 7-12 URLs
 - Is Site Too Slow?
- **SpotCheck Test Service**
 - Do Selected Parts of URLs Have Constant Content?
 - 6 Spots/URL on 6 URLs
 - Anything Missing on Site?
- **InwardLink Test Service**
 - Weekly Update of Inward Link List
Max of 25 Inward Links
 - Do Outsiders' Links to the Site Work?
 - Which Prior InwardLinks Have Disappeared?



eValid -- The Internet Quality Authority



eValid SERVICE OPTONS -- E-Commerce Validation

- **Transaction Test Service**
 - Three Stage Transaction
 - Selection
 - Purchase
 - Payment
 - Does an E-Commerce Transaction Succeed?
- **Security Test Service**
 - Two-State Tests
 - Login Passes Valid User
 - Login Blocks Invalid User
 - Is Login Security Working?



eValid SERVICE OPTONS -- Self-Test Development

- eValid TEST (Test Engine / Standard Tests)
- FREE to eValid Customer
- Records / Edits / Plays
- Submits Keysave File to eValid
 - Internal Confirmation
 - Production Version



eValid -- The Internet Quality Authority



eValid SERVICE OPTONS -- Reporting

- Email Alerts
 - All Errors
 - Warning Counts Exeed Threshod
- Online WebSite Access
 - ErrorLog
 - EventLog Details
- TestSuite Execution
 - Semi-hourly (15 Minutes) 3 Days
 - Hourly 1 Week
 - Four-Hourly 1 Month



eValid -- The Internet Quality Authority



**Software Research, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA**

Phone: [1] (415) 550 - 3020

FAX: [1] (415) 550 - 3030

Email: sales@soft.com

WebSite: <http://www.soft.com>



eValid -- The Internet Quality Authority



eCommerce Benchmarking Methodology and Criteria

Steven Rabin,
Chief Technologist



How Much of Your eCommerce Sales are at Risk?

- 44.1 million on-line shoppers in the U.S.
- 37.5 million more will go on-line in the next 12 months
- on-line buyers spend an average of \$200/month
- an estimated \$762 million/month in eCommerce sales is lost due to slow, spotty, inefficient sites

Zona Research, 7/99

INTERWORLD



Commerce Server Site Analysis

Home Page Size Bail-Out Rates

| | |
|---------|-----|
| 70 KB * | 50% |
| 40 KB * | 30% |
| 34 KB | 8% |

* other pages on site were 32-35KB and showed a 6-8% bail-out rate

vendors must heed the 8 second rule



Keynote Systems, 6/99



Actions Taken After Abandoning eCommerce Site

| <u>Demographic</u> | <u>Did Not Buy Item</u> | <u>Bought Item at Other Web Site</u> | <u>Bought Item at Brick and Mortar</u> |
|--------------------|-------------------------|--------------------------------------|--|
| All | 34% | 24% | 37% |
| Age < 25 years | 27% | 23% | 53% |
| Age 25 to 34 | 43% | 25% | 30% |
| Age 35 to 44 | 27% | 24% | 43% |
| Age 45 to 54 | 35% | 21% | 38% |

Zona Research, 7/99





The eCommerce Performance Life-Cycle

**P
L
A
N**

Define your terms
Understand your customers
Set your goals
Plan your infrastructure

**B
U
I
L
D**

Build and tune your application
Benchmark your system

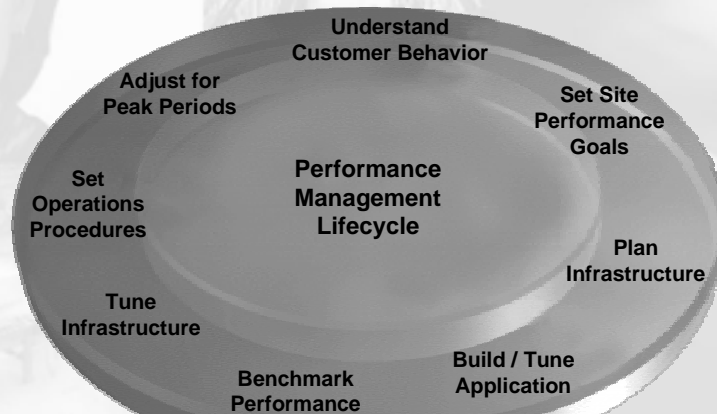
**G
O**

Build operations procedures
Adjust for special events

INTERWORLD



Performance Management Lifecycle



INTERWORLD

Define Your Terms



- **Simultaneous users? Orders / day? Pages / second? Visitors / day?**
- **What is the difference between a page? Hit? Impression?**
- **How complex are the pages?**
- **Are system capacity numbers measured in pages / machine or pages / CPU?**
- **Understand units of commerce work**
 - loading modules
 - building templates
 - round trips (ie. database)
 - transaction types

INTERWORLD

Understand Your Customers



- You can't predict your traffic unless you understand your customers.
- You must predict how many will come and what they will do.
- Rely on past experience, industry statistics, and all information available.
- Consider season, time of day, marketing campaigns, and other factors.

INTERWORLD



Set your goals Sizing the System by Orders / Day

- Orders / Day 1000 [GOAL]
- Conversion Ratio 2% [ESTIMATE]
- Pages / Visit 10 [ESTIMATE]
- Peak Ratio 10% [ESTIMATE]
- Pages / Peak Hour 50000 [CALCULATED]
- Pages / Second 14 [CALCULATED]
- Pages / Second / CPU 2 [IW ESTIMATE]
- CPU's Needed 7 [CALCULATED]

INTERWORLD



Set your goals Sizing the System by Simultaneous Users

- Simultaneous Shoppers 1000 [GOAL]
- Desired Response Time 5 [ESTIMATE]
- Dwell Time 45 [ESTIMATE]
- Pages / Second 20 [CALCULATED]
- Pages / Second / CPU 2 [IW ESTIMATE]
- CPU's Needed 10 [CALCULATED]

INTERWORLD



Build and Tune Your Application

- Budget time for performance tuning
- Understand the trade-off between performance and functionality
- Consider impacts on other systems
- Focus on areas that will be the hardest hit by customers.
- Small decisions can have a huge impact

INTERWORLD



Benchmark Your System

- Benchmark each component in isolation.
- Benchmark the system as a whole.
- Invest the time in accurately modeling your users with a load testing tool.
- Assemble a team of experts to help with the final acceptance test.
- Measure the system against the original goals.

INTERWORLD



Performance & Stability Testing (1/2)

- Simultaneous Users:
Threaded commerce software guards against serialized resources. However, that requires careful management of common services to avoid deadlock. Test for concurrent usage.
 - database connection pools
 - CPU
 - components/objects

- Transactions per second:
Stress testing for peak performance. Turnaway capabilities based on estimated peak availability.
 - common HTML refusal page
 - where is the knee of the curve

INTERWORLD



Performance & Stability Testing (2/2)

- Server Sizing:
 - Memory/CPU's?
 - What is running on each server?
- Database Sizing:
 - Capacity planning using sample data.
 - Foot-print per user.
 - Trigger threshold notices.
- Real-world Simulations:
 - Server log analysis
 - Classify types of transactions. Example: Search versus Orders?

INTERWORLD



Top Commerce Site Performance (Xmas '98)

| <u>Site</u> | <u>HomePage Delivery (seconds)</u> | <u>Availability (percent)</u> |
|-------------------|--|-----------------------------------|
| Amazon | 4.9 | 99.4 |
| Barnes & Noble | 7.3 | 99.2 |
| Dell | 8.3 | 99.1 |
| Gateway | 6.4 | 86.7 |
| Lands End | 9.0 | 91.5 |
| Toys R Us | 5.7 | 98.1 |
| Business 40 index | 9.2 | 97.5 |

INTERWORLD



Benchmark Configurations

| | | |
|-------------|---|---|
| Scenario 1: | <u>Search to Buy Ratio</u> 99 : 1 | <u>Static to Dynamic Pages</u> 1 : 99 5 : 95 10 : 90 |
| Scenario 2: | <u>Search to Buy Ratio</u> 95 : 5 | <u>Static to Dynamic Pages</u> 1 : 99 5 : 95 10 : 90 |
| Scenario 3: | <u>Search to Browse to Buy Ratio</u> 60 : 39 : 1 | <u>Static to Dynamic Pages</u> 1 : 99 5 : 95 10 : 90 |
| Scenario 4: | <u>Search to Browse to Buy Ratio</u> 50 : 45 : 5 | <u>Static to Dynamic Pages</u> 1 : 99 5 : 95 10 : 90 |

INTERWORLD



eCommerce Benchmark Transactions

- Home Page Transaction
- Product Page Transaction
- Section List Transaction
- Shopping Basket Transaction
- Buy Transaction
- User Registration Transaction
- Search Transaction
- Check Status Transaction

Static and Dynamic Units of Work - Simple and Complex Transactions

INTERWORLD



Transaction Type Percent Frequency (bookseller)

- Search on product characteristics (e.g. title 43%)
- Display particular product 32%
- View Homepage 10%
- Display Shopping Cart Contents 8%
- Add Item to Shopping Cart 5%
- Buy a product 1%
- Register new user .3%
- Display order .3%
- Error .3%

INTERWORLD



Benchmark Scripts (Search-Buy)

Home Page

Search

Basket

Order Process

| | | |
|-------|--------------------------|-------------------------|
| OP1 = | ObjBuilder, OrderProcess | initialize |
| OP2 = | " " | verifyOrderDialog |
| OP3 = | " " | verifyBillingDialog |
| OP4 = | " " | verifyShippingDialog |
| OP5 = | " " | setShippingMethodDialog |
| OP6 = | " " | confirmationDialog |
| OP7 = | " " | paymentMethodDialog |
| OP8 = | " " | orderCompleteDialog |
| OP9 = | " " | deleteOrder |

INTERWORLD



Web Site Performance Q&A

- Do customers experience consistently good performance from your web site?
- How does your web site compare to your competitors or to industry benchmarks?
- Do customers in certain cities have more trouble with performance than from other cities?
- How does your site deal with heavy traffic and/or peak load periods?
- How reliable is your site in terms of connections refused, connection time outs and page time outs?
- Are certain pages consistently slow?
- Should geographic mirroring be considered for your site?

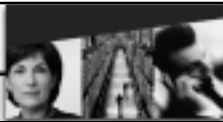
INTERWORLD



Benchmark and Performance Technology Assessment

- **Site Characteristics**
 - Determine customer value of each feature
 - Remove non-essential steps
 - Dynamic page analysis
 - Turn off features (based on peak load scenarios)
 - Remove or disable personalized pages (peak load scenarios)
 - load characterization, site characterization
- **Assessment**
 - Determine customer expectations and evaluate the impact
 - Who are you selling to and what do they expect to see next?
 - Does the current infrastructure support the future?
 - Cost and benefit analysis of new features vs. performance
- **Simulate Real World Load**
 - Number of users
 - Types of transactions
 - Database access
 - Legacy connection(s)
 - Network/infrastructure issues

INTERWORLD



| Element | 1 user load | 5 user load | 10 user load | 20 user load | 50 user load |
|-----------------------|-------------|-------------|--------------|--------------|--------------|
| Home Page | | | | | |
| CatalogList | | | | | |
| SectionList - Cached | | | | | |
| CatProduct - Cached | | | | | |
| SectionPage - Cached | | | | | |
| ProductPage - Cached | | | | | |
| Basket (IW X) | | | | | |
| BasketSteps | | | | | |
| OrderProcess (totals) | | | | | |
| - Initial Step | | | | | |
| - Verify Order | | | | | |
| - Verify Billing | | | | | |
| - Verify Shipping | | | | | |
| - Shipping Method | | | | | |
| - Confirmation | | | | | |
| - Payment Method | | | | | |
| - Order Complete | | | | | |
| - Delete Order | | | | | |
| ObjectBuilder | | | | | |
| - NOP process | | | | | |
| - Static Template | | | | | |
| - Static Template + | | | | | |
| - Database access | | | | | |
| Search | | | | | |
| Category Search | | | | | |
| Customer Registration | | | | | |
| Mix 85% to 15% | | | | | |
| Mix 95% to 5% | | | | | |

Results vary based on site customizations

Use benchmarks as a guide. Your mileage may vary!

Units = Dynamic Pages/sec (transactions/sec)

INTERWORLD

eCommerce Benchmarking Methodology and Criteria





Effective Testing for Java-based Web Software

Jeff Schuster
jschuster@rational.com

Agenda

- ◆ **What is Java software, anyhow?**
- ◆ How do you test Java?
- ◆ How do you build testable Java applets/applications?
- ◆ Discussion

Rational
unifying software teams

Rational
unifying software teams

Understanding Java

- ◆ It is used in many different places
 - Java Applets -- in browsers, applet-viewer
 - Java Applications -- Sun and Microsoft VMs
 - Java Runtime Environment (from Sun)

WRITE ONCE, RUN EVERYWHERE!

Rational
unifying software teams

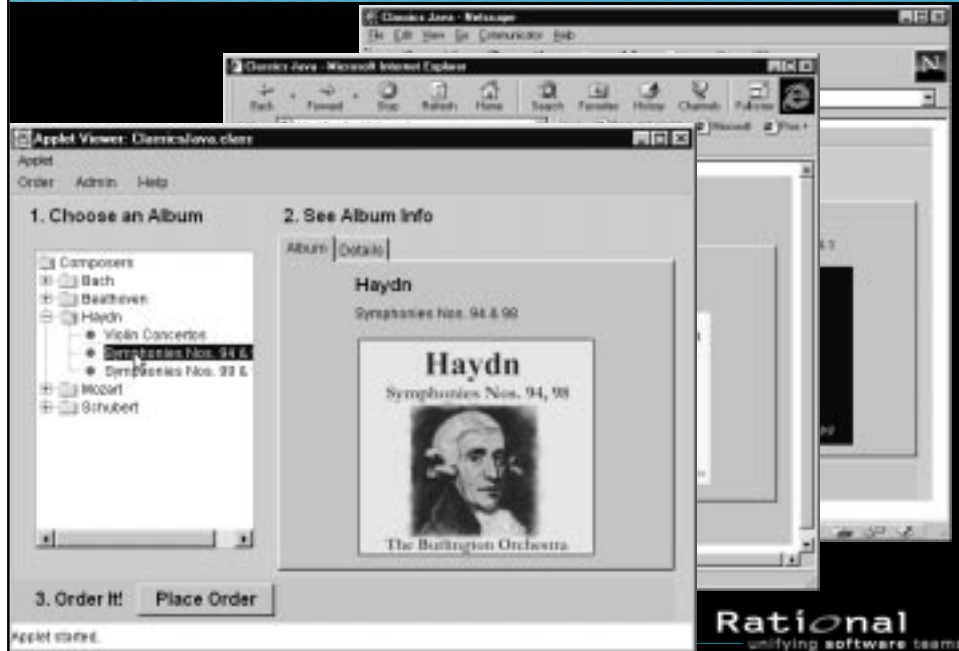
Testing Java

- ◆ It is used in many different places
 - Java Applets -- in browsers, applet-viewer
 - Java Applications -- Sun and Microsoft VMs
 - Java Runtime Environment (from Sun)

WRITE ONCE, TEST CAREFULLY!

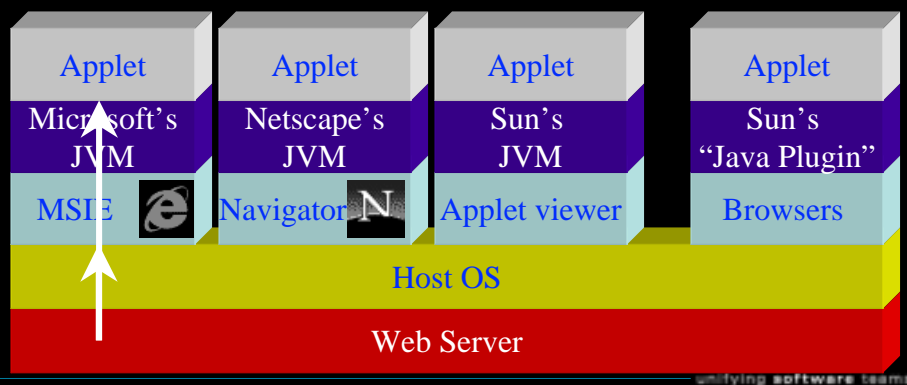
Rational
unifying software teams

Navigator, Explorer, Applet Viewer



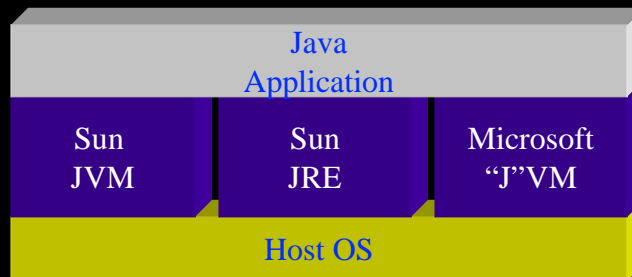
Applets - Web apps running in a browser

- ◆ *Many pieces to this configuration*
- ◆ *Before testing, know the target environments*
- ◆ *Regression testing is necessary*



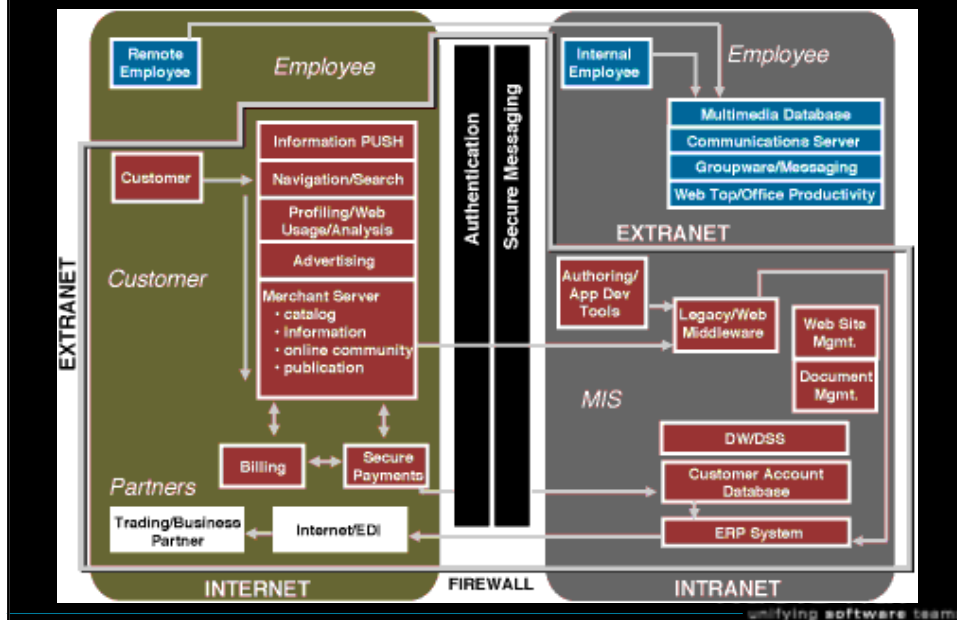
Java Apps - programs on the desktop

- ◆ Several possible environments.
- ◆ Each application is likely to be deployed in just one (or maybe two) environments on Windows



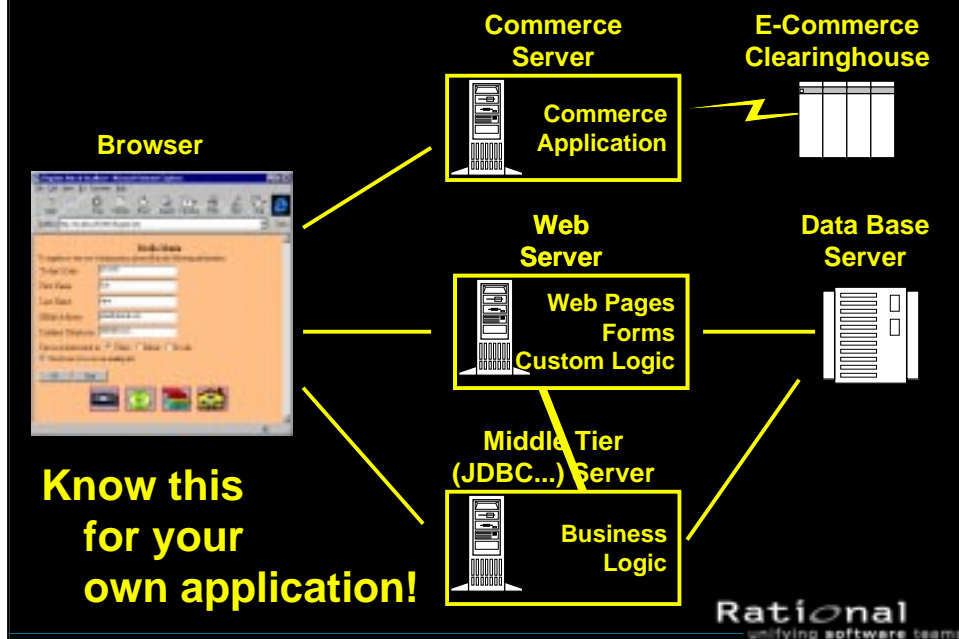
Rational
unifying software teams

Typical E-Commerce Application Structure



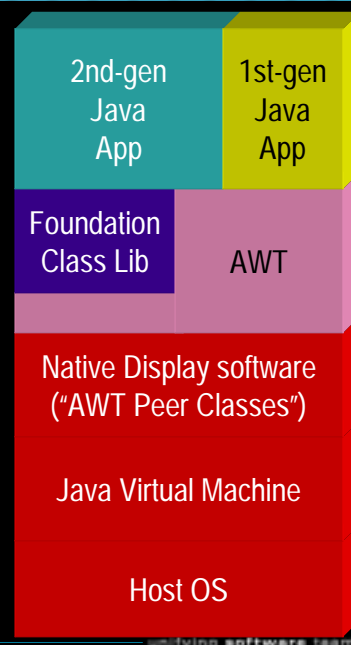
Rational
unifying software teams

Typical Internet Application Structure

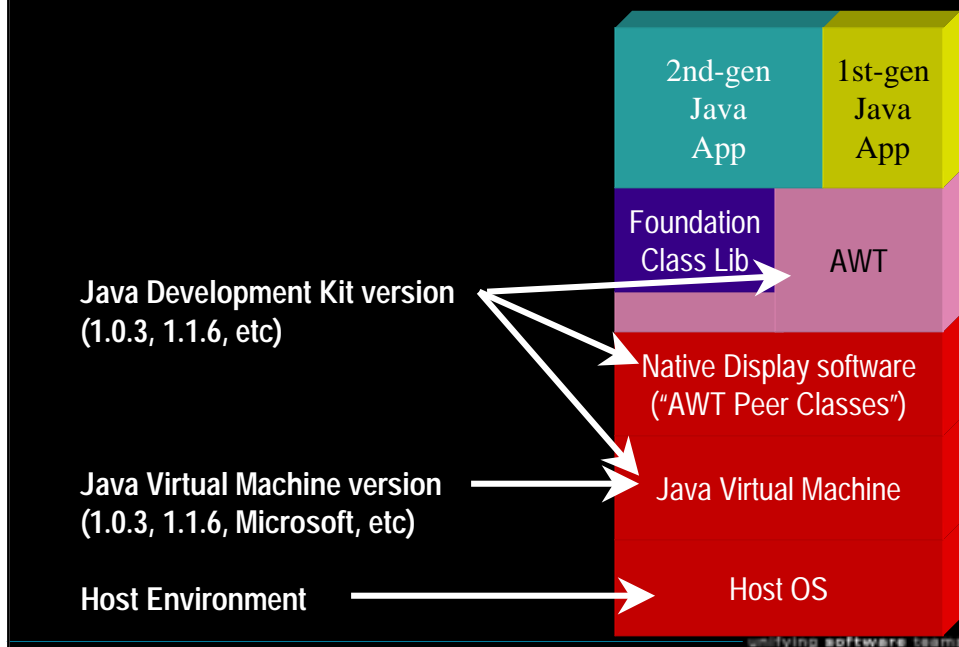


Java software -- what it's made of ?

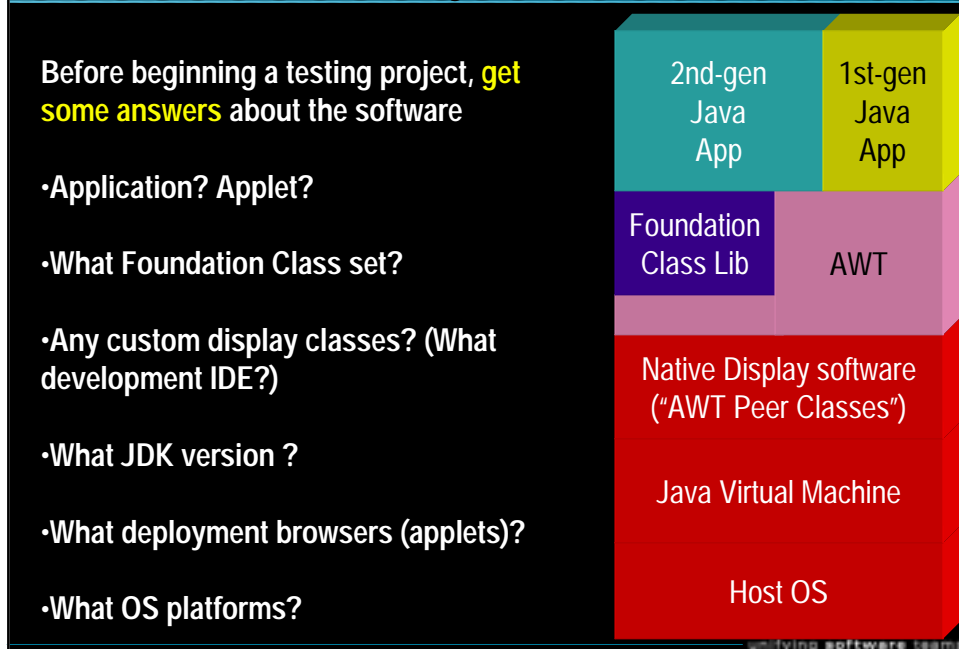
- ◆ GUI toolkit, drawing software
- ◆ Foundation Class choices:
 - Custom classes
 - Sun's JFC / Swing
 - IBM Visual Age/Java
 - Symantec Café
 - MS AFC or WFC
- Abstract Windowing Toolkit underneath other display classes



Java software -- what it's made of ?



Java software -- what you have to know!



Agenda

- ◆ What is Java software, anyhow?
- ◆ **How do you test Java?**
- ◆ How do you build testable Java applets/applications?
- ◆ Discussion

Rational
unifying software teams

Recall what you have to communicate...

- ◆ Requirements
- ◆ Test Plan and coverage
- ◆ Test Scripts
- ◆ Defects - including state, priority, system config., etc.
- ◆ Regression results

The screenshot displays two overlapping windows from the Rational software. The top window, titled 'Requirements Hierarchy', shows a tree structure of requirements for 'TC522 Loans and Database Integrity'. The bottom window, titled 'Rational LogViewer', shows a table of test log events.

| LogEvent | Result | Date | Time | Created | Computer |
|--|--------|----------|-------------|---------|-----------|
| Indicator Pass (ObjectData) ObjectData | Pass | 12/18/99 | 10:23:25 PM | | EDWARDS01 |
| Indicator Pass (ObjectData) ObjectData | Pass | 12/18/99 | 10:23:40 PM | | EDWARDS01 |
| Indicator Pass (ObjectProperties) Object P | Pass | 12/18/99 | 10:23:59 PM | | EDWARDS01 |
| Script (script code) | | 12/18/99 | 10:23:59 PM | | EDWARDS01 |

Are we ready to release?

Rational
unifying software teams

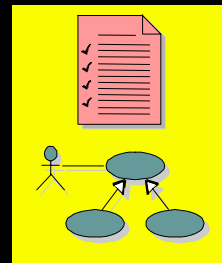
To effectively test Web Apps, you MUST

- ◆ Be able to tell how far we had tested against the **current** system **requirements** in the test plan
- ◆ Check for **reliability** automatically with each new build
- ◆ Create test scripts for new functionality **quickly and easily**
- ◆ Make test scripts on one iteration and **keep using** them on all of them
- ◆ Use one set of test scripts for **all configurations**
- ◆ **Load test** with confidence and ease **before deploying**
- ◆ **Communicate** the results and **share** the test assets and metrics

Rational
unifying software teams

Track dynamic requirements

- ◆ Understand the requirements to create the test
- ◆ Report coverage against **current** requirements
 - Where do results still apply?
- ◆ Track and document

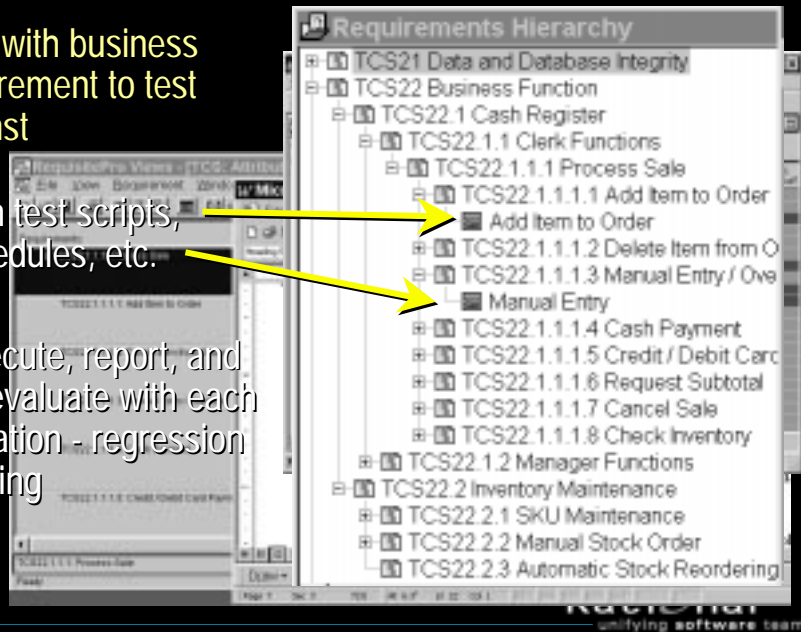


Are we ready to release?

Rational
unifying software teams

Creating the Test Plan

- ◆ Start with business requirement to test against
- ◆ Plan test scripts, schedules, etc.
- ◆ Execute, report, and re-evaluate with each iteration - regression testing



Steps in Functional Regressions Testing

1. Create test scripts

- Select requirement
- Record actions
- Insert validation
- Edit script (optional)

2. Run tests

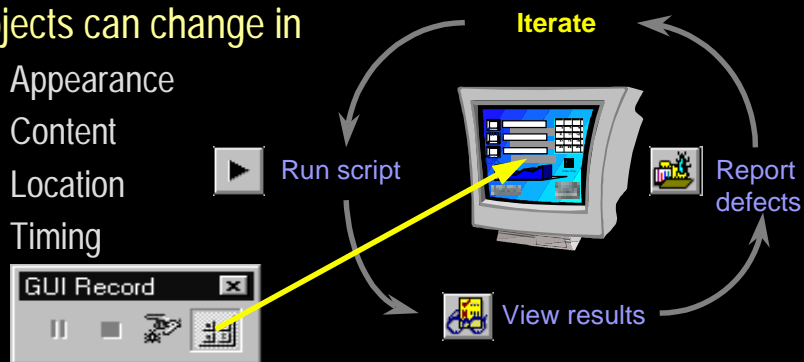


Rational
unifying software teams

Track System-Under-Test Changes

- ◆ Objects can change in

- Appearance
- Content
- Location
- Timing



- ◆ Object recording required to make test scripts robust

- When application changes
- Across configurations

Rational
unifying software teams

Object Testing on Java

Accurate, reliable navigation on Java objects (even tabs, trees, ...)

Tests all data, even hidden (e.g. in Sun Swing JTables)

All object properties available to developer are tested at runtime

| PRICE | CD | MP3 | MP3 | CD | MP3 | CD | MP3 | CD | MP3 |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 12.99 | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |

HTML Requires Object Testing Too

The screenshot shows the Amazon.com homepage in Microsoft Internet Explorer. A tool window titled "Object Properties Verification Point" is overlaid on the page. Red circles and arrows indicate the following:

- A red circle around the "Book recommendations" link in the "Books" section, with an arrow pointing to the tool's "Name" field containing "HTMLLinkHTMLLinkbook-recommendations".
- A red circle around the "Book recommendations" link in the "MUSIC" section, with an arrow pointing to the tool's "Name" field containing "HTMLLinkHTMLLinkbook-recommendations".
- A red circle around the "http://www.amazon.com/..." URL in the tool's "URL" field.

Yellow text annotations on the right side of the screenshot state:

- Server generates unique target for every transaction
- Can not record targets
- Must have persistent link names and let server use real targets at runtime
- Object-oriented recording does it right, based on W3C Document Object Model.

Rational
unifying software teams

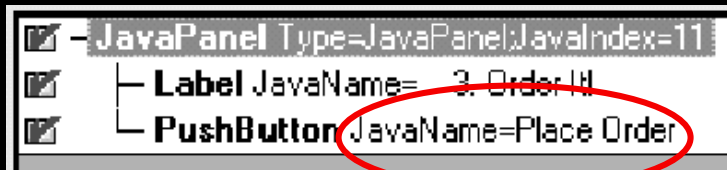
Agenda

- ◆ What is Java software, anyhow?
- ◆ How do you test Java?
- ◆ **How do you build testable Java applets/applications?**
- ◆ Discussion

Rational
unifying software teams

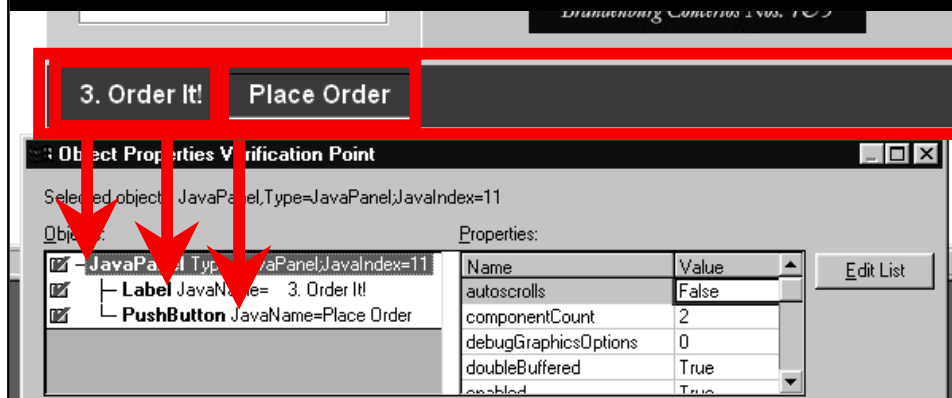
Give Objects Persistent Names

- ◆ Set the **name** member variable -- optional --
 - Increases testability by allowing object recognition by name.
 - For example, in AWT:
orderButton.name = ".Place Order";
 - ... in JFC:
orderButton.accessibleName = "Place Order";



Flatten the Hierarchy and Make Names Unique

- ◆ Naming of Java objects: flatten out the subdivision hierarchy; do not depend on inheritance
 - E.g. "JavaPanel; JavaIndex=11" include Label & PushButton



Use Standard Object Classes

- ◆ Derive your custom Java object classes from standard JFC / AWT classes
 - A good GUI testing tool recognizes the object as an instance of the standard object from which it is derived
- ◆ Put testability information into public member variables in your Java objects
 - A good GUI testing tool sees these public members as object properties.

Rational
unifying software teams

Building Testable Java Applets/Applications?

Three simple rules

- ◆ Give Objects Persistent Names
- ◆ Flatten the Hierarchy and Make Names Unique
- ◆ Use Standard Object Classes; Expose Public Members

Rational
unifying software teams

Agenda

- ◆ What is Java software, anyhow?
- ◆ How do you test Java?
- ◆ How do you build testable Java applets/applications?
- ◆ Discussion

Rational
unifying software teams

Writing a Software Engineering Handbook for the Intranet

Benedikt Lutz

Siemens AG Austria, PSE BV QM

1 Summary

In this paper I describe the Software Engineering Method of Siemens AG Austria PSE, which is an intranet-based application of about 1000 html-pages. The intranet has many practical advantages for such solutions (direct access to information in the working environment, platform independent, no distribution effort,..), but for the practical acceptance of voluminous online handbooks the usability of the application is extremely important: The whole text has to be rewritten/"redesigned" for online presentation, the architecture of the application must have a clear structure (defined and recognizable page types and link types), the access structure for different needs of the users has to be carefully designed.

In this paper the basic principles of the method as well as the realization of the intranet application will be shown; in the presentation (paper 10B) I give a live demonstration of the application.

2 Short presentation of Siemens AG Austria – PSE

PSE (Program and Systems Engineering) is a part of the Siemens Aktiengesellschaft Österreich. PSE offers solutions and services for the entire information and communications technology sector. About 3.900 engineers, more than half of which hold an university degree, work in Vienna, Graz, Salzburg and 5 international locations. More than 30,000 man-years of experience have been gathered in the field of communication systems and data processing. Around 95% of the services offered by PSE have been exported to 53 countries up to now.

PSE is at home working with most system platforms and operating systems available on the market. The comprehensive overview of the current and future potential in public and private communication networks provides a basis for special competence with respect to the design and implementation of integrated information systems – from PC networks to satellite communications.

PSE has been concentrating its efforts for many years now on quality-assured development techniques. The ISO 9001 certificate awarded to PSE only constitutes the starting point for further improvements. SEI assessments, conducted in accordance with the capability maturity model, have resulted in PSE assuming a leading position among the cream of the international competition.

3 SEM® - the Software Engineering Method of PSE

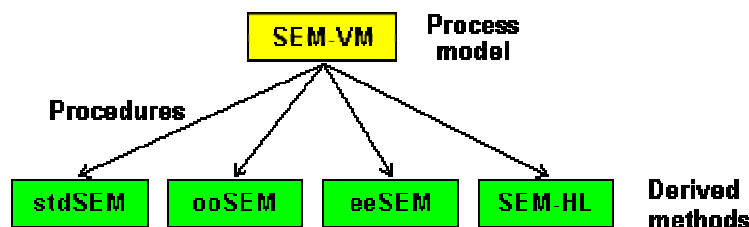
SEM is a process model which defines all work and test steps and the results which are to be achieved in all phases of system development. It is therefore fully compliant with the standard ISO 9001. The SEM development method was devised as early as 1983 for the PSE and has since been the subject of systematic training in seminars (basic training in software engineering for all employees).

Since it was first developed, the method has been adapted several times to meet market requirements. As early as 1993, SEM was used as a basis for certifying the entire PSE in compliance with the standard ISO 9001. One year later, the PSE software development was assessed using the CMM/Bootstrap maturity model. This assessment rated SEM as a „robust, comprehensive process model”.

For PSE, the systematic use of SEM brings about a marked improvement in error detection in the early development phases and thus makes a significant contribution to business success.

Since 1996 we redesigned our engineering method completely: We started with writing an general process model *SEM-VM* (German: „Vorgehensmodell”), which defines a uniform and abstract process (like a „law book”) and serves as a reference model for certifications (like ISO 9001 or SEI assessments). From this abstract process model there can be derived tailor-made method descriptions (or instances), which are adapted to requirements specific to a particular sector or technology. So the process model for the user can be much more concrete and helpful than the abstract description.¹ Until now, there were derived the following instances:

- *stdSEM*: a general model for system development
- *ooSEM*: development of systems based on object-orientation methods
- *eeSEM*: system development method for developing electronics, firmware and ASICs
- *SEM-HL*: development and / or procurement of software for the Semiconductor Group (HL) of Siemens AG Berlin / Munich.
- Other instances, like SEM for product development (*prodSEM*), are in preparation.

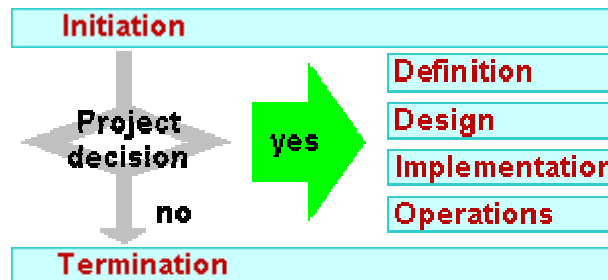


picture 1: The abstract process model and derived methods

SEM subdivides the development process in phases. According to SEM, a project begins in the *Initiation* phase and always ends in the *Termination* phase. If a positive project decision is reached in the Initiation phase (after all risks have been evaluated and after basic advance planning), the project is continued with the *Definition* phase; e.g. for drawing up a tender.

¹ The whole process model, the procedures how to derive concrete methods and the description of our whole project appeared as a book: Hermann Kaindl/Benedikt Lutz/Peter Tippold: Methodik der Softwareentwicklung. Vorgehensmodell und State-of-the-Art der professionellen Praxis. Vieweg Professional Computing 1998.

The detailed project planning, processing of requirements and producing of the software requirements specification are followed by the *Design* and *Implementation* of the ready-to-use product, accompanied by the necessary project management and quality management measures. Product utilization itself is ensured in the *Operations* phase by means of agreed accompanying measures. In the *Termination* phase, every project is concluded in an orderly manner and experience gathered during the project is recorded in a report.



picture 2: Project phases according to SEM

But SEM does not only support the traditional *Waterfall model*, which is still important for „classical“ software development work. In addition, there are possible 4 other life cycle approaches:

- The *Evolutionary development model* serves as a basis for performing maintenance projects and version developments in the PSE: A follow-up product version is always based on the preceding version.
- In the *Prototyping model*, design and implementation are performed jointly step-by-step using suitable tools: This ensures that customers' needs are taken into account more than ever before in the development work (this life cycle approach is very similar to the so called rapid application development in some other publications).
- In the *Incremental delivery model*, the architectural design specification serves as a basis for further development in independent releases.
- The *Spiral model* is a model involving extensive accompanying measures which is used with very large projects (for each phase: goal definition, risk analysis, simulations, etc.).

4 Quality management in the PSE

Quality management has always been attached particular importance in the PSE:

- Every project has a *QA manager* who is responsible for QA planning, QA reporting and ensuring that all QA measures are performed correctly.
- Every division has its own *quality manager* who supports the divisional manager in attaining the business goals.

- The *quality management center* advises the PSE management in formulating and implementing Q policy. The center also supports the various divisions and international departments in PSE-wide improvement activities, the efficiency of which is regularly checked. Technical and commercial project controlling procedures have been defined and introduced to ensure the success of projects.

In addition to SEM, the PSE has numerous project support tools. These include:

- *Guides* (Performance of reviews, initiation of projects, risk management, usability engineering, archiving of documents, etc.)
- *Evaluation tables* (Ascertaining project risks, ascertaining project manager qualifications, etc.)
- *Programming guidelines*

Support centers are available for important subject areas. These centers provide advice and support throughout the PSE. Support centers currently exist for:

- Estimation of effort and metrics
- Configuration management
- Databases
- Test and test automation
- Object-oriented development
- Project experience
- Project management
- Windows support & solutions
- Components and internet technology

5 From printed process models to online documentation

Printed process descriptions in the field of software development have always been seen by most of the developers as a „necessary evil“, which has to be obeyed: People do not like „paperware“ that is difficult to read and, moreover – at least according to the opinion of many developers - restricts their „liberty and creativeness“.

Online documentation, on the other hand, is much more attractive for developers, especially in the form of an intranet application:

- An electronic manual in HTML format can be called up online from every workstation and has a uniform appearance under all operating systems.
- The intranet contains a wealth of useful information for efficient networking.
- Documents can be downloaded directly from the online display of the application.
- Printouts can be performed from every workstation.

- There is no organizational effort involved in distributing new versions; the intranet Web is always up-to-date.

So it was clear for us that the new version of SEM had to appear as an online application.

In many companies the existing paper-based documentation is already put into the intranet (e.g. circulars, organigrams, work specifications, instructions etc.). In most cases the existing text is just converted 1:1 into html-format. This procedure is absolutely sufficient for short documents with clear access structures (e.g. table of contents of all circulars of the company with title and date; if you click the title, the circular is shown in html-format or downloaded as a WinWord-file).

But if you handle more voluminous documents this way, there arise big problems, which have to do with reader acceptance and applicability of the texts. Nobody reads many online pages in one piece, people lose control over the whole text. The two most important problems in this context are *disorientation* of the reader (the so-called „lost in hyperspace“-phenomenon) and *cognitive overload* (too much information which I do not need in the concrete context). These questions have been treated from the scientific point of view for over a decade in the hypertext-community, in Information Science, Cognitive Psychology and Text Linguistics; the discussion moved into the web in the last years: There is a huge mass of literature available on the usability and accessibility of texts in the web; hypertext styleguides are *en vogue*.

The essence of all this scientific research and advising literature: Hypertexts have to be *designed* in a structural, textual and graphical sense to be applicable and usable for the reader. An important aspect is the definition of *node types* (content of the same category) to enhance the recognition of the readers; and on the other hand the definition of *link types* (leading to nodes of a specified category) to give the right „presentiment“ and safety to the readers („where will I go to, if I click this icon?“).

These findings make it very clear: If you plan to construct a voluminous online handbook which is usable and accepted by the readers, the application has to be carefully *designed*, already existing linear text has to be reworked - or even better - *completely reformulated*. Early versions of the application have to be *usability tested* (to check if your imaginations also work in reality).

6 Design principles and architecture

To design the architecture of a online handbook, you have to keep in mind on one side the *contents* which have to be conveyed, and on the other side the *needs of the users* (what has to be noticed by the readers under any circumstances vs. what are the most important needs of the reader). You just cannot command the user to read this or that html-page; you have to design the web in such a way that the probability is as high as possible that readers will read and understand the most important pages. In the case of an instruction manual this task is even more difficult: People have not only to understand the contents, you have to lead people to *act* in the right way!

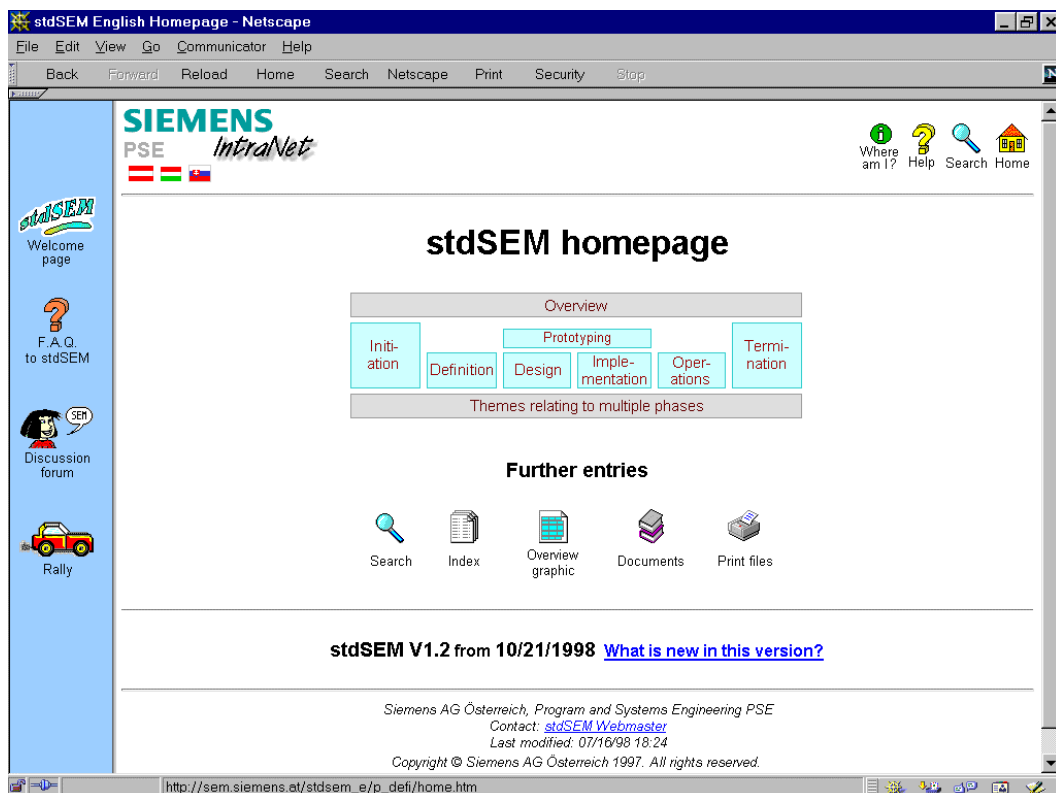
This means for the structure of such a web: Clear content structures with clearly identifiable and accessible contents. People must know what is in the web and what not (the „borders of the universe“).

SEM is a phase model; so it was clear for us that the dominant structure for description and access had to be phase-oriented. That is why the most important parts of SEM are in the

description of the development phases. In addition, there are some themes which are not clearly assigned to specific phases, but belong to *multiple phases* (like project management and control, quality assurance, configuration management, different life cycle approaches,...). These themes appear in all phases, but are described coherently in a more global context („Themes relating to multiple phases”). Finally we needed an introductory overview and some surrounding amenities (help system, discussion forum for stimulating discussion on SW engineering themes,...).

So our handbook consists of the following parts:

- In-depth description of development phases
- Themes relating to multiple phases
- Overview
- Help system
- Discussion forum
- Rally (for making people curious to know what is in the web)
- Furthermore, there are short introductions to the contents and handling of the application (for beginners).



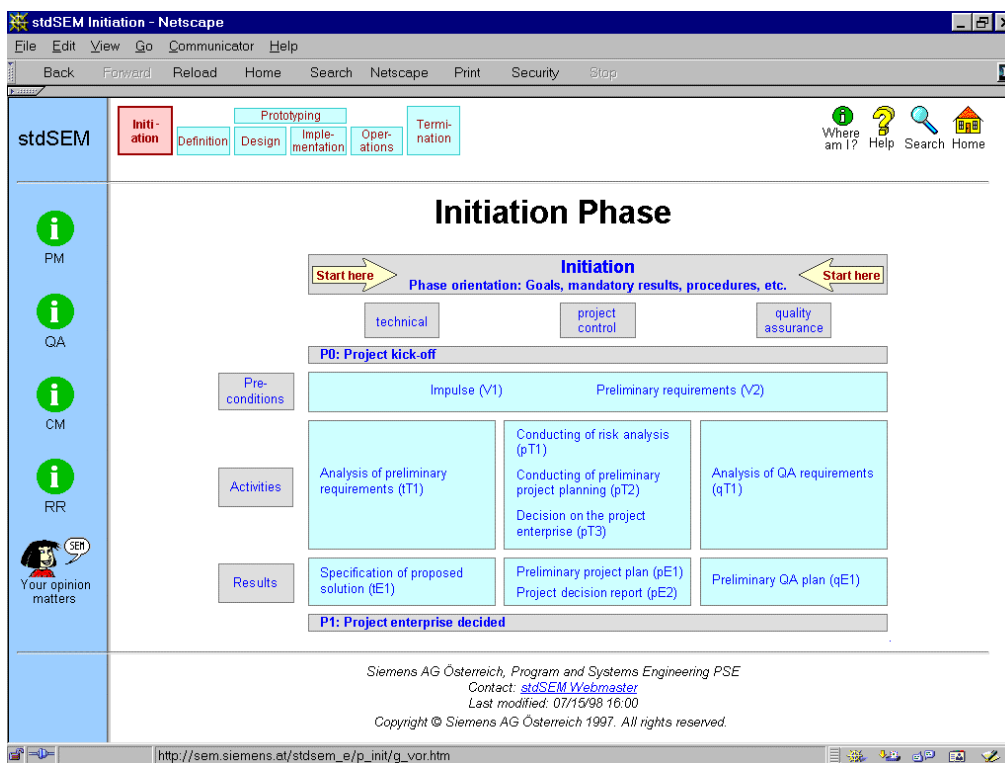
picture 3: stdSEM homepage

Phases according to SEM are characterized by specific *goals* to be reached. These goals are reached by executing activities and the results which these yield. For the majority of phases in stdSEM, related activities are grouped together into appropriate *subphases*. These subphases should be executed if the associated activities make sense (e.g. subphase „Drawing up a tender“ in the Definition phase, subphase „Preparation of operations“ in the Implementation phase). The subphases can also overlap.

Every *phase* has its required *preconditions*. The phase can only be started if these preconditions are met. There are also required *activities* which are performed during the phase. stdSEM makes a distinction between primarily *technical, project control and quality assurance* activities. Each phase is characterized by *results* which arise when activities are executed. The degree of obligation of these results is *defined* for each phase in stdSEM (in the „Phase Orientation“).

Milestones mark significant points during the course of the project which are generally associated with important results. Since they are particularly suitable for *project controlling, prespecified milestones are defined* in stdSEM (these can also be modified on a project-specific basis). For example, „Project enterprise decided“ is a project control milestone in the Definition phase.

In these phase orientation pages, which give a general graphical overview of the phase, there are many links to detailed descriptions: each activity and result can be clicked to get in-depth information; the general goals, milestones and phase-specific aspects of phase independent themes (like project management) can be called up as well:

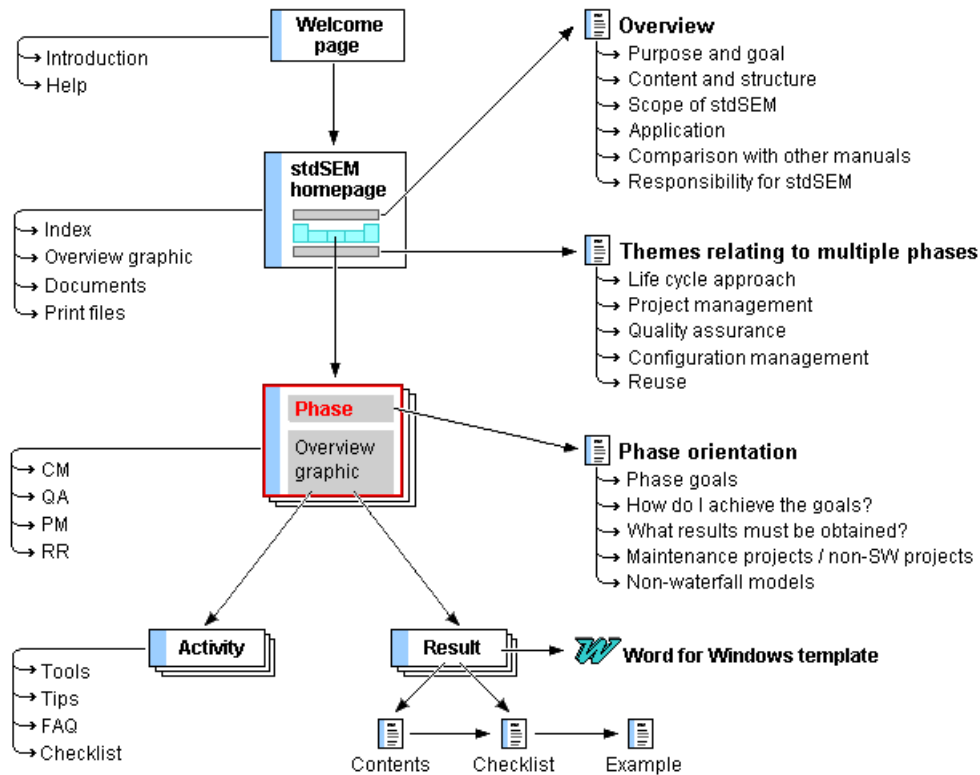


picture 4: Phase homepage

From the description of activities and results there are „deepening links” to further information: checklists, tips, references to tools,...

For documents you will find predefined templates in different formats (WinWord and FrameMaker), checklists and sample documents.

So the overall structure of stdSEM looks as follows:



picture 5: Architecture of the whole stdSEM-Web

7 Text design and layout of pages

Reading from the screen differs from reading printed matter. This has to be taken into account when you write online texts and if you intend that the text is really read by the readers. But out of many scientific publications of different disciplines it is not quite clear, what this means in detail: For „hard” empirical scientific research there are too many intervening variables (like screen size and resolution, font type, font size and color, text formatting, background color, combination of graphic elements with text, system reaction time, etc.; all this aside from the most important reader variables and the use of different browsing systems). So the results of scientific research in this field are either too specific or very general. Furthermore, in the last years practice developed much faster than theory, if you look for example at the development of the WWW (in the technical, textual and graphical sense).

Nevertheless, some principles for the writing of online texts can be postulated: The text must be shorter and more concise than printed text, there must be more headings and subheadings (for catching attention), there must be short paragraphs and a clear structure of the text, important text parts have to be highlighted. The „message” has to arrive at the reader in the first seconds of reading, people do not continue reading on the next screen if they are not quite sure that they find relevant and interesting information at that place.

In this sense, these principles have quite a lot of similarities with journalism, especially boulevard journalism: The text has to „fight for attention”. The classical structure of newspaper articles is in the form of the so-called „inverted triangle”: Headline, lead sentence, important content, less important content (you can shorten the article from the end without loss of relevant information and consistency, if necessary). I think, the writers of online texts can learn much from the procedures in journalism.

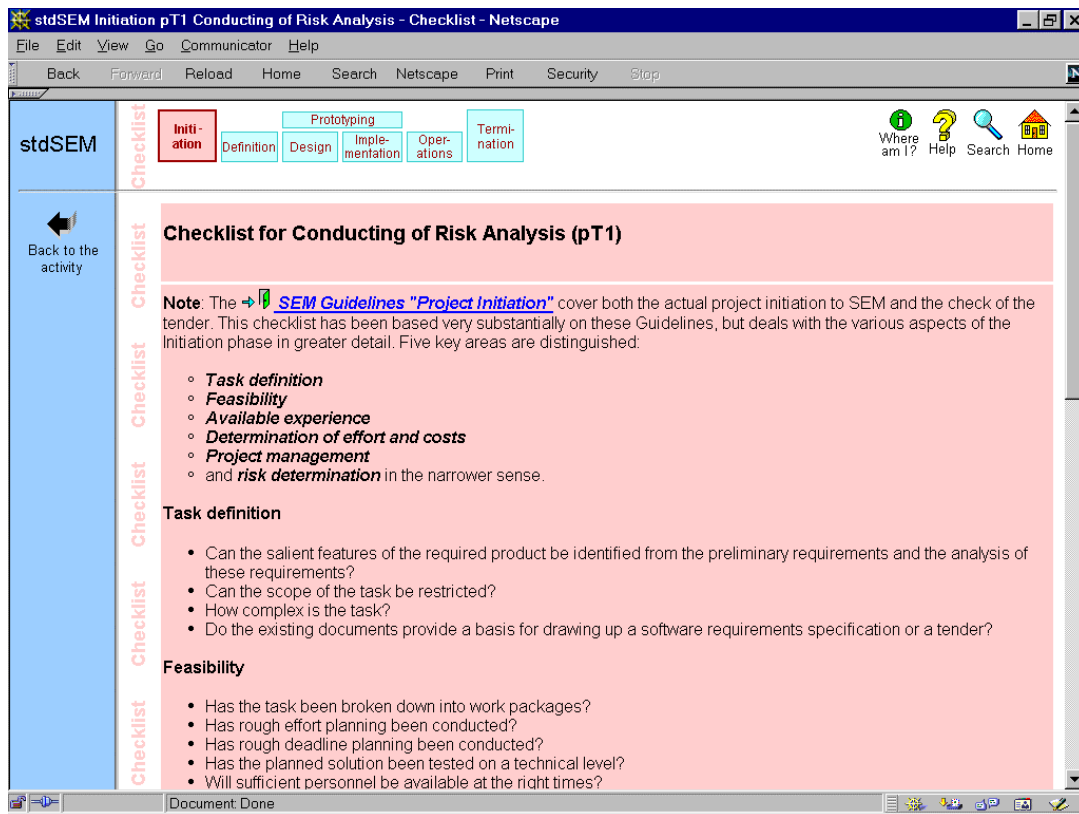
This holds true also for the combination of textual and graphical elements: The reading process is lead and structured strongly by the use of graphical elements. Graphical cues are very important and effective for orientation, recognition, signaling of importance and visualization of context. Therefore we use graphic elements extensively in stdSEM. When we usability-tested an early version of stdSEM, we made the interesting observation that sometimes the effect of graphic elements is so strong that people did not realize what to do, if some procedures are documented only in a textual way – the text-only descriptions actually „drowned” and were overruled by the text-and-graphics descriptions. So we had to design the whole web graphically.

Now I want to present some typical pages (node types) of stdSEM for better understanding of these principles. The next picture shows an activity node. Such nodes contain always a clickable image map, where the graphic shows the connection to the previous and following activities („where do I come from” – „what other activities are in connection with this activity” – „where shall I go to”).

The text structure of *activity nodes* is always the same: to be performed by - goal of the activity – short description of activity – further notes. In the margin of the page there may be links to in-depth descriptions of the activity (like checklists, tips, references to tools and FAQ-pages). Sometimes there are links to the „outer world” of the intranet (to pages not belonging to the stdSEM web): these links are signaled by specific icons (WinWord-symbol, door symbol):

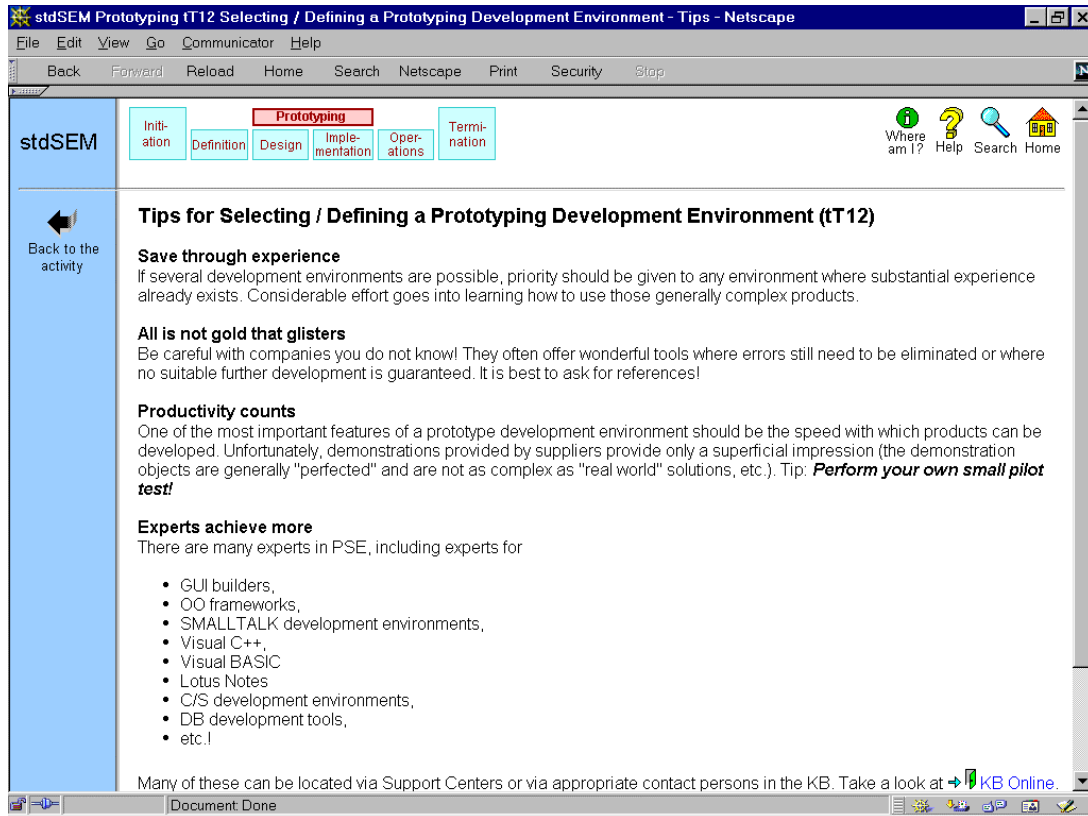
picture 6: Activity node

Checklists give more details to an activity and are structured in the form of a catalogue of questions and detailed descriptions of the steps belonging to this activity. They are identifiable in the web through a pink background color:



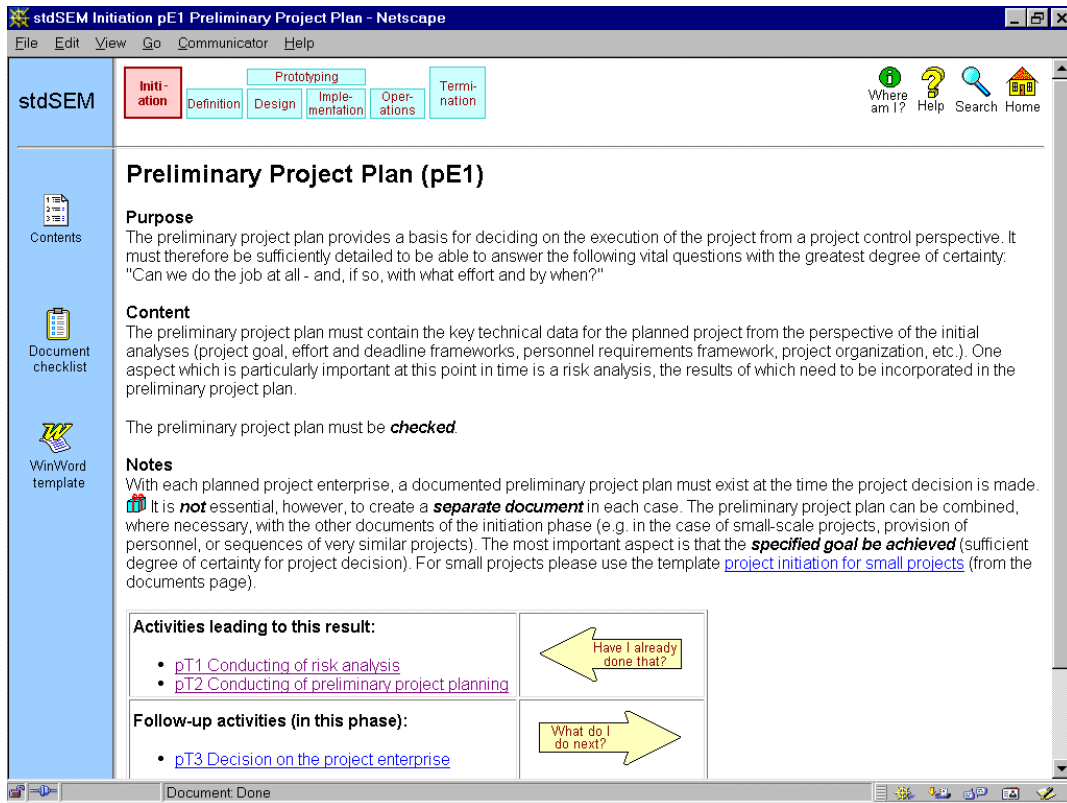
picture 7: Checklist for activity

Pages with tips have a different style: They are less „serious” and give „real life advice”; people are addressed directly in these pages:



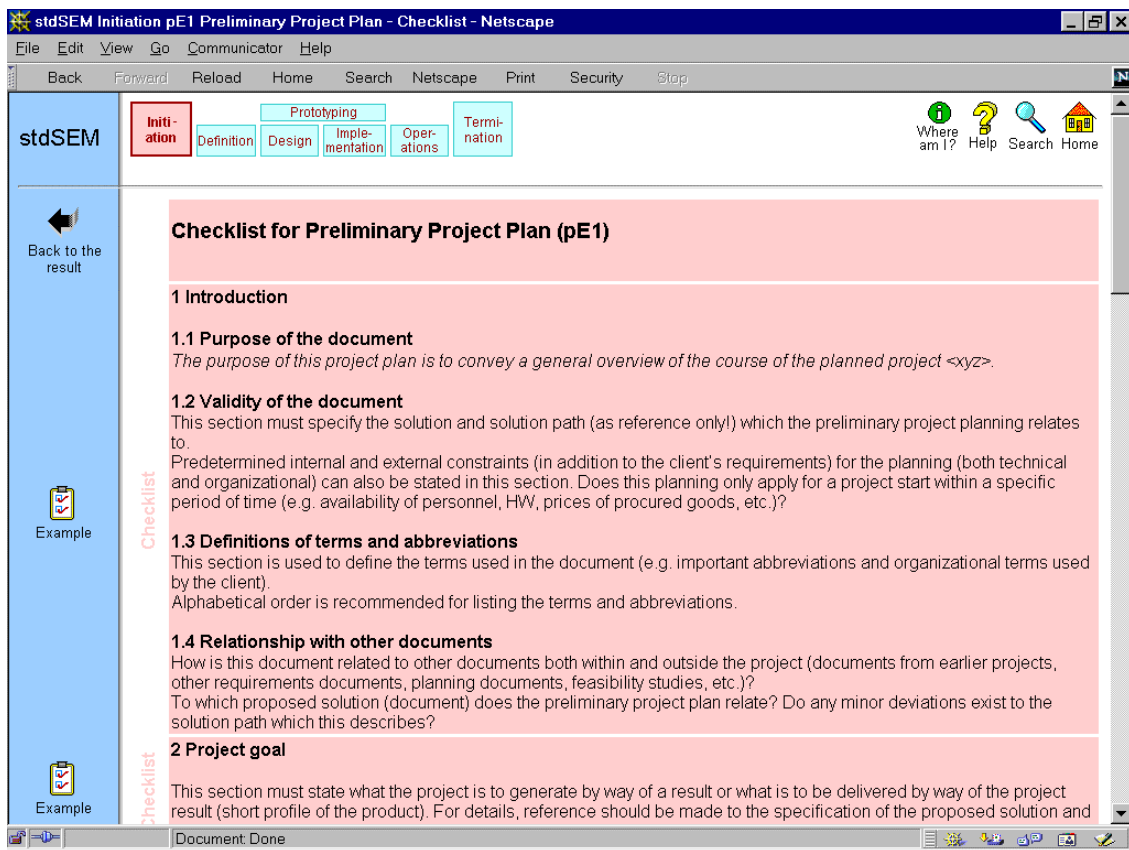
picture 8: Tips for activity

Result nodes describe the result to be reached (in many cases the results are documents). The description follows a general structure: Purpose – short description of content – further notes – graphical description of preceding and following activities. The most important links at the result nodes lead (in the case of documents) to a table of contents of the document, to the document checklist (the annotated table of contents), and to the download of the document template:



picture 9: Description of result

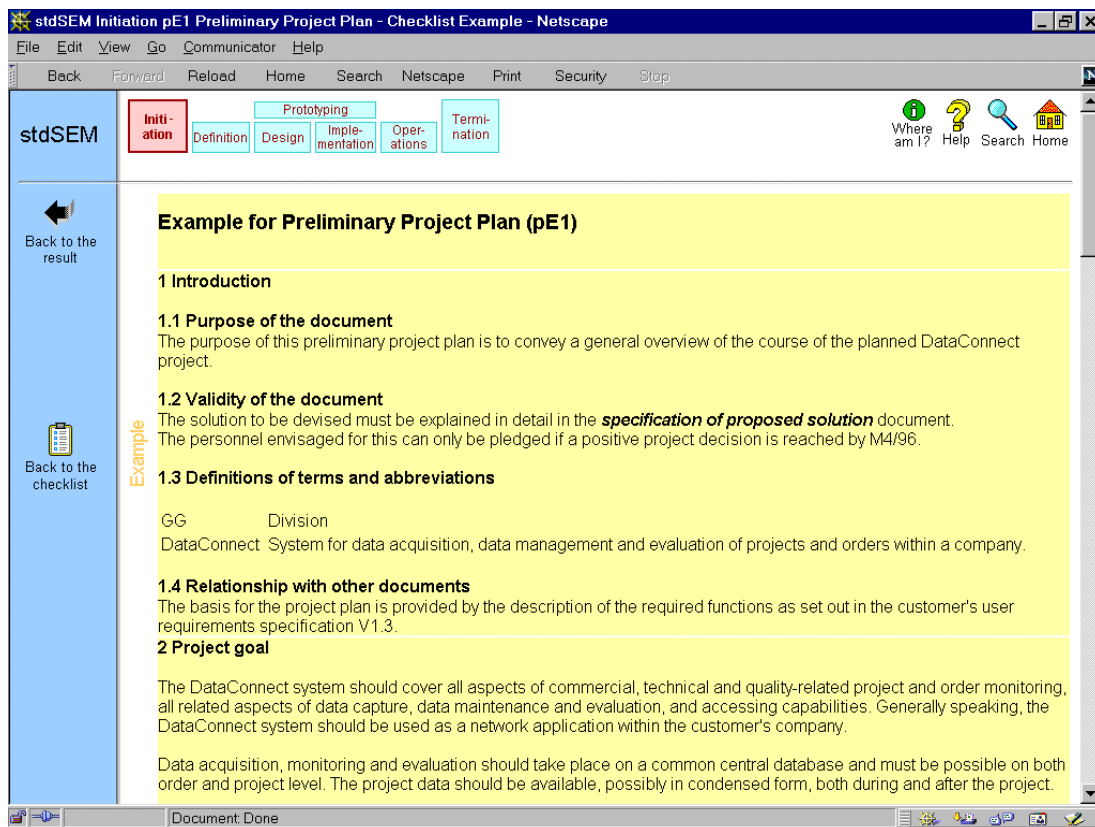
The annotated table of contents has the advantage for the user, that there are clear structures for the document to be written and, moreover, help and advice in an argumentative manner, what has to be written in this or that chapter and why:



picture 10: Checklist for result

From this annotated table of contents you can click to sample documents (which are only available in html-format). These sample documents are characterized in the web through a yellow background color. From cognitive psychology we know that „learning by example” is very effective, but dangerous, on the other side: The examples are perceived so dominantly that it is difficult to understand the underlying and general principles (the examples are „seductive”; people think that it will go always the way how is described here).

That was the most important reason why we decided to offer the document templates as annotated tables of content and not as sample documents (they are too seductive): If there is already an almost finished document, people tend to copy given materials and do not think much about if this or that description holds true for the concrete project, too. Moreover, it is fairly probable that people forget important themes, if these themes are not mentioned in the sample document (and you cannot mention every possibly relevant theme in a sample document).



picture 11: Sample document

8 Orientation and multiple access structures for the readers

Apart from a clear structure and typed links and nodes, it is very important in voluminous websites to provide orientation and different access structures to the readers, according to their different needs.

Orientation is supported through a general header on every page with links to the homepage, the phase homepages (the current phase is highlighted), to a search engine, a help system and a „where am I”-page, which dynamically shows your actual position in a map of the stdSEM-web (see picture 5).

For the fast access to the contents and definitions of central terms we provide an (alphabetical) *index* with links to the respective pages (this index is *content-oriented* and not automatically generated, which was much work, but leads to good results).

In the web there is integrated a *search engine* with full text search possibilities.

From the homepage there is direct access to some pages which lead to *often needed content* (see picture 3): There is the possibility for downloading all document templates in different formats from one single page (for the „power user”), and links to the descriptions of results and print-files (see next chapter). Fast access is very important for experienced users, which is a fact that we could also confirm through the analysis of logfiles.

9 Still a printed version?

If online information consists of more than small and independent pieces (like in databases), there will sooner or later arise the question: Is there no printed version of the whole content or specific „chapters” available? – For us, this question was already important during the writing process: It is very tedious to review online text; it is much more effective to read the text in a printout and to mark the sections to be changed in handwriting (you have to read and review not only single pages, but pages in the whole context).

But there are some principal problems in printing websites: Hypertext is from its nature *nonlinear* (there are no clearly preceding and following pages to a specific page, which in fact is the important „informational surplus” of hypertexts), but you can *print* only in a *linear form*. A linearized print is less attractive than the online presentation, but on the other hand it is sometimes necessary (and the users asked for it).

So we decided to offer printfiles for download from the web (we put pages in a plausible sequence into „chapters” in short and long form, with a table of contents and numbered pages) in the format of PDF- and PostScript files:

stdSEM

Initiation Definition Design Implementation Operations Termination

stdSEM Print Files

If you want to read individual sections of stdSEM in printed form, you can download the appropriate pdf files and PostScript files. You can choose between two versions for each phase:

- The **overview** contains the overview graphic and important pages relating to goals and results.
- The **full version** contains all pages in a phase.

| Theme | Version from | Size in A4 Pages | PDF File | | PostScript File | |
|--|--------------|------------------|------------------------------|-------|-----------------------------|-------|
| | | | Link | MByte | Link | MByte |
| Introduction to contents and use | 21.10.98 | 30 | intro.pdf | 0,4 | intro.ps | 2,0 |
| Help pages | 21.10.98 | 25 | help.pdf | 0,3 | help.ps | 1,0 |
| Overview | 21.10.98 | 23 | overview.pdf | 0,2 | overview.ps | 0,8 |
| Themes relating to multiple phases | 21.10.98 | 94 | themes.pdf | 0,8 | themes.ps | 3,0 |
| Overview graphics of the phases and general overview | 21.10.98 | 9 | gra.pdf | 0,3 | gra.ps | 1,8 |
| Initiation overview | 21.10.98 | 22 | init1.pdf | 0,2 | init1.ps | 0,8 |
| Initiation | 21.10.98 | 60 | init2.pdf | 0,5 | init2.ps | 2,0 |
| Definition overview | 21.10.98 | 68 | defi1.pdf | 0,6 | defi1.ps | 1,9 |
| Definition | 21.10.98 | 229 | defi2.pdf | 1,9 | defi2.ps | 7,3 |
| Design overview | 21.10.98 | 57 | desi1.pdf | 0,5 | desi1.ps | 1,7 |
| Design | 21.10.98 | 180 | desi2.pdf | 1,5 | desi2.ps | 6,0 |
| Implementation overview | 21.10.98 | 42 | impl1.pdf | 0,4 | impl1.ps | 1,4 |
| Implementation | 21.10.98 | 144 | impl2.pdf | 1,3 | impl2.ps | 5,2 |
| Prototyping overview | 21.10.98 | 43 | prot1.pdf | 0,4 | prot1.ps | 1,4 |
| Prototyping | 21.10.98 | 169 | prot2.pdf | 1,5 | prot2.ps | 6,1 |
| Operations overview | 21.10.98 | 22 | oper1.pdf | 0,2 | oper1.ps | 0,9 |
| Operations | 21.10.98 | 97 | oper2.pdf | 0,9 | oper2.ps | 3,5 |
| Termination overview | 21.10.98 | 21 | term1.pdf | 0,2 | term1.ps | 0,7 |

picture 12: Print-files in PDF and PostScript

10 The course of this project / lessons learned

At first we thought that we could use text parts of the already existing and well introduced printed software engineering handbook and then add the important new themes (like prototyping and new technologies). But as we tried to convert some existing text parts into html-format, then it became clear, that we had to start from the very beginning, if we wanted good results which are accepted and usable for our end users.

So we made a complete design and architecture of the online application (as shown above); we started with small prototypes of phases and performed usability tests which gave valuable input for the design of the next prototypes.

Parallel to these design-oriented actions we defined and formulated the *content*, based on already existing procedures (especially the old written handbook and the abstract process model SEM-VM) and the results of a working group of representatives of different business units of PSE. We started with the work results (documents) and went over to the description of the activities leading to these results. Finally we formulated the checklist and tip pages.

A good *WYSIWYG-authoring* tool is indispensable for work in such a project (collaborative authoring must be supported); the writers have to leave the level of html completely, they must concentrate on the contents using predefined templates of pages giving already a structure for the text and page design. The writing process for hypertext authors is very demanding, because you have always to think on different levels („what do I want to write, which node is the best place for this or that information, did I already mention this aspect in a neighbor node, what links have to be set from this node to other nodes”, etc.).

Usability inspections (fast design cycles) and usability testing are extremely important for such projects. You need *real project leaders* as „guinea pigs” in such usability tests to check the real expectations and problems of real life; otherwise the results are not relevant enough. The usefulness of the application in everyday projects is the crucial point for such tests. Furthermore you have to check some „simple usability themes” as well (like design and comprehensibility of icons etc.).

The *introduction process* of new methods is very important for the use in everyday life (and is worth the investment). It is not enough to just mail to everybody that the new method is now available. At least you need company-wide presentations to everybody and special introduction for middle management. In addition, we had a very successful online competition (rally), where people were led through the web by solving puzzles and could win a prize.

If you once attract the attention of the users, you have to keep them interested. So the *maintenance* of the website is very important (and expensive – don't forget budgeting of maintenance!). You have to develop the method constantly, give feedback to the users, make the concrete support to the projects better, according to the needs of the users of this method. All this is much easier to perform with an online software engineering method than with traditional printed handbooks.

Writing a Software Engineering Handbook for the Intranet

Dr. Benedikt Lutz
Siemens AG Austria, PSE BV QM

Overview

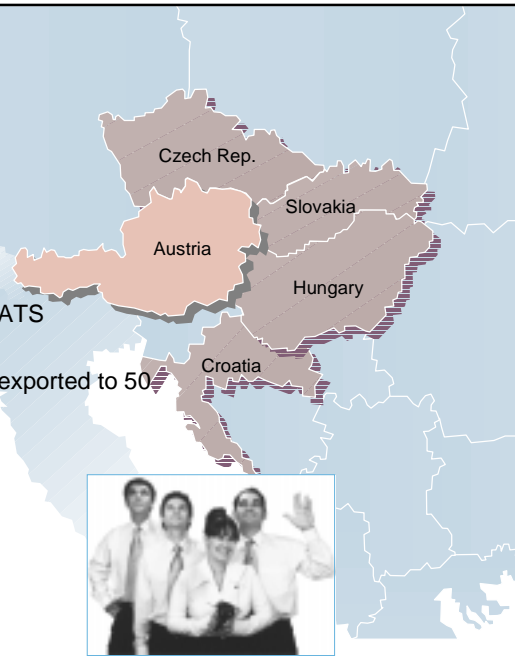
- Short presentation of Siemens AG Austria - PSE
- Quality Management in the PSE
- SEM[®] - The Software Engineering Method of PSE
- From printed process models to online documentation
- Description of stdSEM[®]
- Important issues of the stdSEM[®] -project
- Live presentation of stdSEM[®]

Program and Systems Engineering PSE

- Is a part of the Siemens Aktiengesellschaft Österreich
- Offers solutions and services for the entire information and communications technology sector
- Is working with most system platforms and operating systems
- Is a provider of services for Siemens worldwide companies and a few external customers
- Sets for itself the highest standards as far as product and process know-how are concerned

PSE: Key Figures

- About 3,900 employees
- 5 company locations in Europe and the USA
- Total output about 4.9 billion ATS (FY 97/98)
- About 92.5% of the output is exported to 50 countries worldwide



QM Organization in the PSE

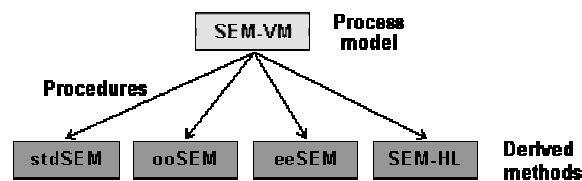
- **QA manager** in every project (responsible for QA planning, reporting, ensuring that all QA measures are performed correctly)
- **Quality manager** in every division (supports the divisional manager in attaining the business goals)
- **Quality management center** (advises in implementing Q policy, supports improvement activities, project controlling procedures,...)

SEM[®] - The Software Engineering Method of PSE

- The first version of the process model SEM was devised already in 1983
- SEM was the basis for certifying PSE in compliance with the Standard ISO 9001 in the year 1993
- CMM/Bootstrap assessments: “robust, comprehensive process model”
- Since 1996 complete redesign and rework of the method: SEM VM, hypertext stdSEM, ooSEM,...

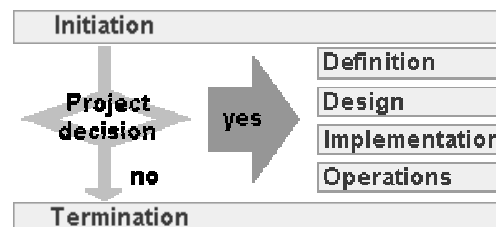
Basics of SEM[®] and stdSEM[®]

- Uniform and abstract process model (“law book” SEM-VM)
- Specific and concrete methods for the users are derived from this abstract model



Course of a project

- Frame phases (initiation/termination) and execution phases
- Project execution is also possible using customer methods



Different life cycle approaches, not only waterfall



Prototyping model



Evolutionary development model



Incremental delivery model



Spiral model

From printed process models to online documentation

- Printed handbooks have always been seen as “necessary evil”
- Online documentation is information in your working environment, “at your fingertips”
- The intranet is platform independent
- New versions are immediately available, no distribution effort
- “SW-Engineering portal” in the company: links to other websites (e.g. QA, Support-centers offering concrete help for special themes and technologies,...)

Hypertext-design

- If printed matter is converted 1:1 into online text, the results are poor; for good results the whole text has to be rewritten
- Two primary problems for the readers:
 - Lost in hyperspace (disorientation)
 - Cognitive overload (too much information)
- ➔ Design and Usability of the Website is very important:
 - Architecture of the Website: Safety, “borders of my universe”
 - Text structure: “inverted triangle” like in newspapers
 - Node-Types: Content of same categories - better recognition
 - Link Types: Links to nodes - giving the right “presentiment”
 - Different access structures, satisfying users’ expectations

Architecture of the stdSEM-web



- about 1000 html-pages
- about 450 graphic elements
- about 17.000 links
- German and English version
- Discussion forum included
- about 35 templates (MS-Word, FrameMaker)
- Embedded into the worldwide Siemens Intranet

Text design for screens

- Reading from screens differs from reading printed matter (about **30% slower reception**)
- Clear arrangement of text parts, **short paragraphs** with headlines are important
- **Inverted triangle** (like in newspapers): headline, lead sentence, important content, less important content
- Combination of text and graphic elements influences the reading process ("**once graphics - ever graphics**")
- Readers scroll only if they are really interested
- In hypertexts the "**cohesive closedness**" of pages is important - you cannot presuppose that the reader already read the "previous page"

Node types

- Content of the same category has to be "**typed**" - better recognition for the reader
- Some important node types of stdSEM:
 - Description of activity
 - Tip page for activity
 - Activity checklist
 - Description of result
 - Annotated checklist for template
- Examples in the live presentation

Access structures

- Different access structures are important for different levels of user knowledge and interest: **“beginner”** - **“browser”** - **“power user”**
- stdSEM supports distinct access structures for **satisfying different needs:**
 - Phase specific home pages
 - Free text search
 - Theme specific index
 - Overview graphic
 - Fast access to all document templates
 - “Where am I” page for supporting orientation



No printed version?

- Already during the writing process there arose the problem of **printing hundreds of pages** (especially for reviewing the content).
- Printing more than a few pages from the web browser is tedious
- A linearized version of a hypertext is less attractive, but in our case necessary (and the users wanted it)
- Our solution: **pdf and ps-files** with linearized “chapters” of the web can be downloaded from the web (incl. table of contents and page numbers)

Special question: What is the best aid for writing a document?

- **No template**, only process-oriented description of activities leading to a result: much liberty, little concrete help
- **Template** with predefined layout, table of contents and chapter headings: help for structuring, but in practice people often do not know what to write in this or that chapter, they just “associate freely”
- **Sample document**: Good, but seductive: People do not change what has to be changed
- We decided for document **templates with annotated chapter headings** (“this section must specify.... It is important to mention..., because...”)

Some lessons learned in this project

- **Usability engineering activities** are extremely important for the acceptance and every day usefulness of the web
 - Design prototypes
 - Usability testing of prototypes
 - Short cycles for design details
 - Analysis of logfile as input for better access structures
- For collaborative authoring a **good WYSIWYG-tool** is important
- **Design and content** are closely connected - we could not strictly separate writing from layout

Summary

- The **intranet** is nowadays the right place for a “software engineering handbook”: Concrete online applicability
- Large webs have to be **designed carefully**: overall architecture, node- and link types
- Already existing texts from printed handbooks have to be **completely rewritten**
- **Multiple access structures** and aids for orientation are necessary
- Design aspects and **usability issues** are crucial for user acceptance (bad usability destroys the best content)

And now...

SIEMENS PSE *IntraNet*

Wagner Austria Österreichs Wirtschaft

stdSEM auf deutsch

Welcome to the hyperworld of stdSEM!

Version 1.2

stdSEM® The software engineering method of PSE for standard projects

Our hint: (important, if this is your first visit or you aren't yet an experienced surfer)

- As an introduction, we offer you the [new contents of stdSEM](#).
- [How to use the stdSEM](#) shows how you can find your way round the web quickly.
- Then take a look at the [plan](#) of the web (icon "Where are IT").
- After this move to the [help page](#) and click on various items to help acquaint yourself with how the help system works.
- You can then (hopefully!) start working without problems ([Homepage](#)). Please use the e-mail address at the foot of each page for [feedback](#), questions and criticisms.

You can return to this Welcome page by clicking the "Welcome" icon in the stdSEM homepage, for example.

The stdSEM team would like to wish you every success with your projects!

Seven Views to Website Quality Modeling and Assessment

Hans-Ludwig Hausen

German National Research Centre for Information Technology
GMD; Schloss Birlinghoven; D-53754 Sankt Augustin; Germany
<http://www.scope.gmd.de> < <ftp://ftp.gmd.de/GMD/SW-Quality>
E-mail: hausen@gmd.de

Quality is to be defined, measured and assessed with respect to
the extent to which stated or implied requirements are met !!!

"When you can measure what you are speaking about, and express it in numbers, you know something about it;
but when you cannot measure it, and when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind;
it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science."

William Thomson, Lord Kelvin, in: Popular Lectures and Addresses, 1891-1894 as Listed in Bartlett's Quotations 1980

Motivation Questions concerning a Website

Topics

- *Motivation*
- *Site Q Attributes*
- *Site Service*
- *Site as SW Entity*
- *Methods*
- *Tools*
- *Objectives*
- *Synthesis*

Site Assessment: -> Needs/Conditions/Conduction
comparing actual measurement results against required

Site Certification: -> Needs/Conditions/Conduction
checking conditions and eventually issuing a certificate

Site Measurement: -> Needs/Conditions/Conduction
mapping of an attribute onto real numbers

Site Validation: -> Needs/Conditions/Conduction
test against implied needs i.e. assumptions

Site Verification: -> Needs/Conditions/Conduction
test against stated needs i.e. specifications

Product:
software comprising at least requirements, specifications and program(s)

Process:
planned, controlled and reported actions to construct, apply or maintain
software product

Project:
planned, controlled and reported process and product

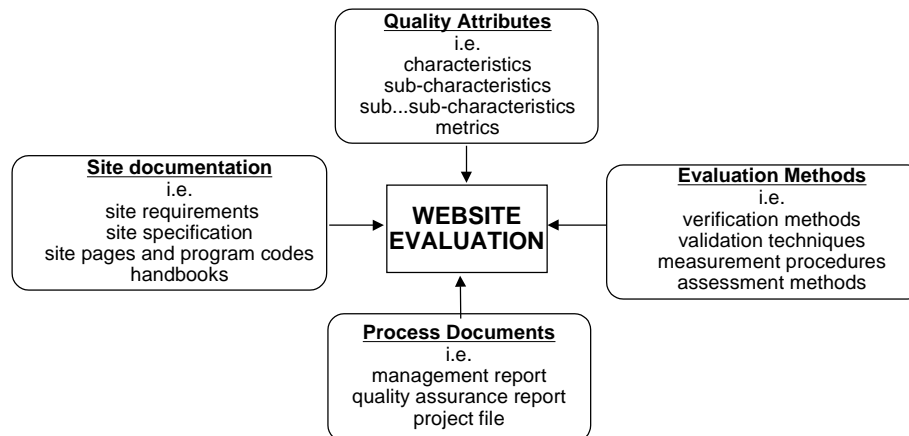
An Engineer's Evaluation Principles

according ISO

- **Repeatability:** Repeated evaluation of the same product to the same evaluation specification by the same testing laboratory gives the same result.
- **Reproducibility:** Repeated evaluation of the same product to the same evaluation specification by different testing laboratories gives the same result.
- **Impartiality:** Evaluation is free from unfair bias towards achieving any particular result.
- **Objectivity:** The evaluation result is obtained with the minimum of subjective judgement.

evaluation = verification + validation + measurement + assessment

Web Site Evaluation Aspects



Q VIEW PERFUM

ISO 9126 Quality Characteristics

Portability A set of attributes that bear on the ability of software to be transferred from one environment to another

Sub-characteristics: adaptability, conformance, installability, replaceability = acir

Efficiency A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used

Sub-characteristics: resource behaviour, time behaviour = rt

Reliability A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time

Sub-characteristics: fault tolerance, maturity, recoverability = fm

Functionality A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs

Sub-characteristics: accurateness, compliance, interoperability, security, suitability = aciss

Usability A set of attributes that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users

Sub-characteristics: learnability, operability, understandability = lou

Maintainability A set of attributes that bear on the effort needed to make specified modifications

Sub-characteristics: analysability, changeability, stability, testability = acst

--->PERFUM-acir-rt-fmr-aciss-lou-acst

NEW - PERFUM - DEFINITIONS

ISO 9126 Quality Characteristics

The followings are examples of proposed definitions.

Functionality: Totality of attributes of software that influence the existence of a set of functions and their specified properties. The functions are those that satisfy stated and implied needs. The totality is expressed by the extent to which software provides functions which meet stated and implied needs when used under specified conditions. (This definition is consistent with ISO 8402.)

Functionality measure: An assigned value which shows the extent to which software provides functions which meet stated and implied needs when used under specified conditions.

Functionality attributes: Those attributes (elementary properties) of a software that influence the existence of a set of functions and their specified property.

Olsina's Quality Characteristics for Academic Sites

1. Usability

- 1.1 Global Site Understandability
 - 1.1.1 Global Organisation Scheme
 - 1.1.1.1 Site Map
 - 1.1.1.2 Table of Content
 - 1.1.1.3 Alphabetical Index
 - 1.1.2 Quality of Labelling System
 - 1.1.3 Student-oriented Guided Tour
 - 1.1.4 Image Map (Campus/Buildings)
- 1.2 On-line Feedback and Help Features
 - 1.2.1 Quality of Help Features
 - 1.2.1.1 Student-oriented Explanatory Help
 - 1.2.1.2 Search Help
 - 1.2.2 Web-site Last Update Indicator
 - 1.2.2.1 Global
 - 1.2.2.2 Scoped (per sub-site or page)
 - 1.2.3 Addresses Directory
 - 1.2.3.1 E-mail Directory
 - 1.2.3.2 Phone-Fax Directory
 - 1.2.3.3 Post mail Directory
 - 1.2.4 FAQ Feature
 - 1.2.5 On-line Feedback
 - 1.2.5.1 Questionnaire Feature
 - 1.2.5.2 Guest Book
 - 1.2.5.3 Comments
 - 1.3 Interface and Aesthetic Features
 - 1.3.1 Cohesiveness by Grouping Main Control Objects
 - 1.3.2 Presentation Permanence and Stability of Main Controls
 - 1.3.2.1 Direct Controls Permanence
 - 1.3.2.2 Indirect Controls Permanence
 - 1.3.2.3 Stability
 - 1.3.3 Style Issues
 - 1.3.3.1 Link Colour Style Uniformity
 - 1.3.3.2 Global Style Uniformity
 - 1.3.3.3 Global Style Guide
 - 1.3.4 Aesthetic Preference
 - 1.4 Miscellaneous Features
 - 1.4.1 Foreign Language Support
 - 1.4.2 What's New Feature
 - 1.4.3 Screen Resolution Indicator

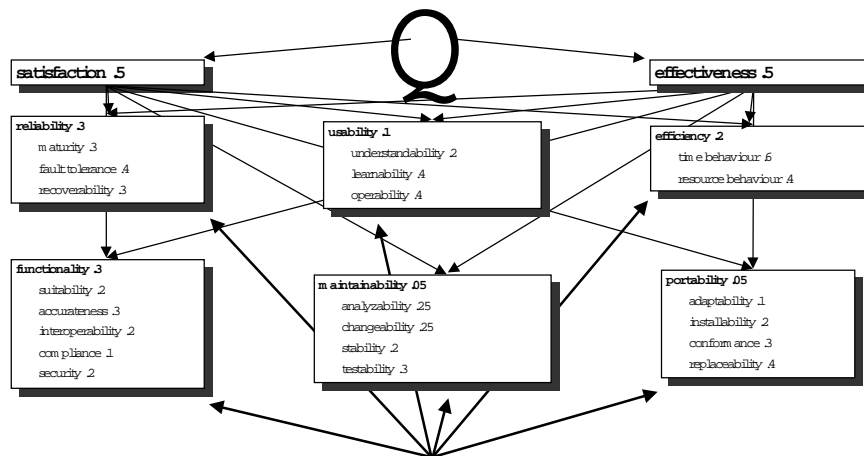
2. Functionality

- 2.1 Searching and Retrieving Issues
 - 2.1.1 Web-site Search Mechanisms
 - 2.1.1.1 Scoped Search
 - 2.1.1.1.1 People Search
 - 2.1.1.1.2 Course Search
 - 2.1.1.1.3 Academic Unit Search
 - 2.1.1.2 Global Search
 - 2.1.2 Retrieve Mechanisms
 - 2.1.2.1 Level of Retrieving Customisation
 - 2.1.2.2 Level of Retrieving Feedback
 - 2.2 Navigation and Browsing Issues
 - 2.2.1 Navigability
 - 2.2.1.1 Orientation
 - 2.2.1.1.1 Indicator of Path
 - 2.2.1.1.2 Label of Current Position
 - 2.2.1.2 Average of Links per Page
 - 2.2.2 Navigational Control Objects
 - 2.2.2.1 Presentation Permanence and Stability of Contextual (sub-site) Controls
 - 2.2.2.1.1 Contextual Controls Permanence
 - 2.2.2.1.2 Contextual Controls Stability
 - 2.2.2.2 Level of Scrolling
 - 2.2.2.2.1 Vertical Scrolling
 - 2.2.2.2.2 Horizontal Scrolling
 - 2.2.3 Navigational Prediction
 - 2.2.3.1 Link Title (link with explanatory help)
 - 2.2.3.2 Quality of Link Phrase
- 2.3 Student-oriented Domain-related Features
 - 2.3.1 Content Relevancy
 - 2.3.1.1 Academic Unit Information
 - 2.3.1.1.1 Academic Unit Index
 - 2.3.1.1.2 Academic Unit Sub-sites
 - 2.3.1.2 Enrolment Information
 - 2.3.1.2.1 Entry Requirement Information
 - 2.3.1.2.2 Form Fill/Download
 - 2.3.1.3 Degree Information
 - 2.3.1.3.1 Degree Index
 - 2.3.1.3.2 Degree Description
 - 2.3.1.3.3 Degree Plan/Course Offering
 - 2.3.1.3.4 Course Description
 - 2.3.1.3.4.1 Comments
 - 2.3.1.3.4.2 Syllabus
 - 2.3.1.3.4.3 Scheduling

- 2.3.1.4 Student Services Information
 - 2.3.1.4.1 Services Index
 - 2.3.1.4.2 Healthcare Information
 - 2.3.1.4.3 Scholarship Information
 - 2.3.1.4.4 Housing Information
 - 2.3.1.4.5 Cultural/Sport Information
- 2.3.1.5 Academic Infrastructure Information
 - 2.3.1.5.1 Library Information
 - 2.3.1.5.2 Laboratory Information
 - 2.3.1.5.3 Research Results Information
- 2.3.2 On-line Services
 - 2.3.2.1 Grade/Fees on-line Information
 - 2.3.2.2 Web Service
 - 2.3.2.3 FTP Service
 - 2.3.2.4 News Group Service
- 3. Site Reliability
 - 3.1 Non-deficiency
 - 3.1.1 Link Errors
 - 3.1.1.1 Broken Links
 - 3.1.1.2 Invalid Links
 - 3.1.1.3 Unimplemented Links
 - 3.1.2 Miscellaneous Errors or Drawbacks
 - 3.1.2.1 Deficiencies or absent features due to different browsers
 - 3.1.2.2 Deficiencies or unexpected results (e.g. non-trapped search errors, frame problems, etc.) independent of browsers
 - 3.1.2.3 Dead-end Web Nodes
 - 3.1.2.4 Destination Nodes (unexpectedly) under Construction
 - 4. Efficiency
 - 4.1 Performance
 - 4.1.1 Static Page Size
 - 4.2 Accessibility
 - 4.2.1 Information Accessibility
 - 4.2.1.1 Support for text-only version
 - 4.2.1.2 Readability by deactivating Browser Image Feature
 - 4.2.1.2.1 Image Title
 - 4.2.1.2.2 Global Readability
 - 4.2 Window Accessibility
 - 4.2.2.1 Number of panes regarding frames
 - 4.2.2.2 Non-frame Version

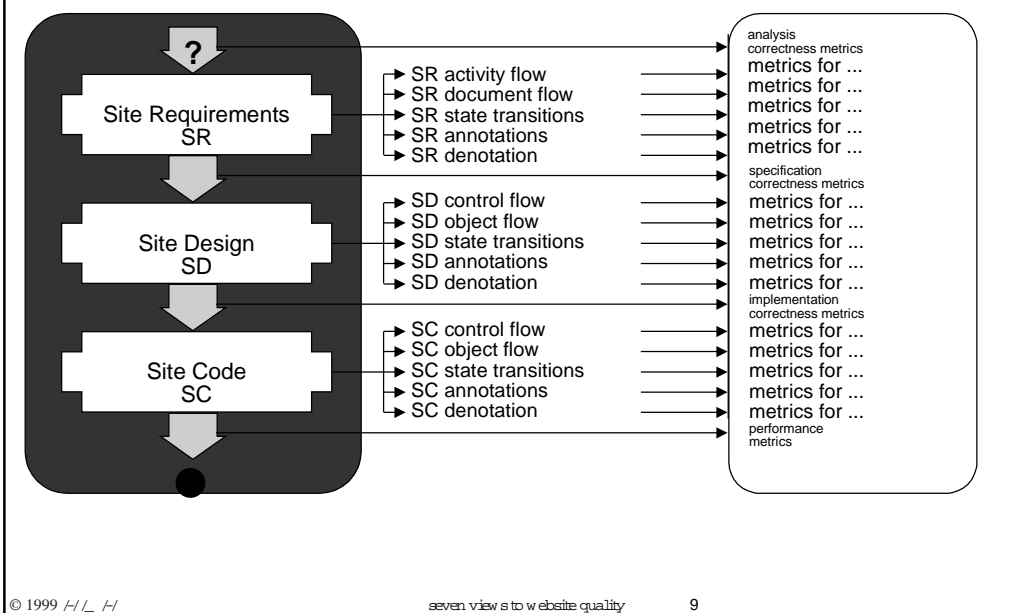
Attributes - Factors - Metrics

(as in ISO/IEC 9126 vs ISO 9241-11)

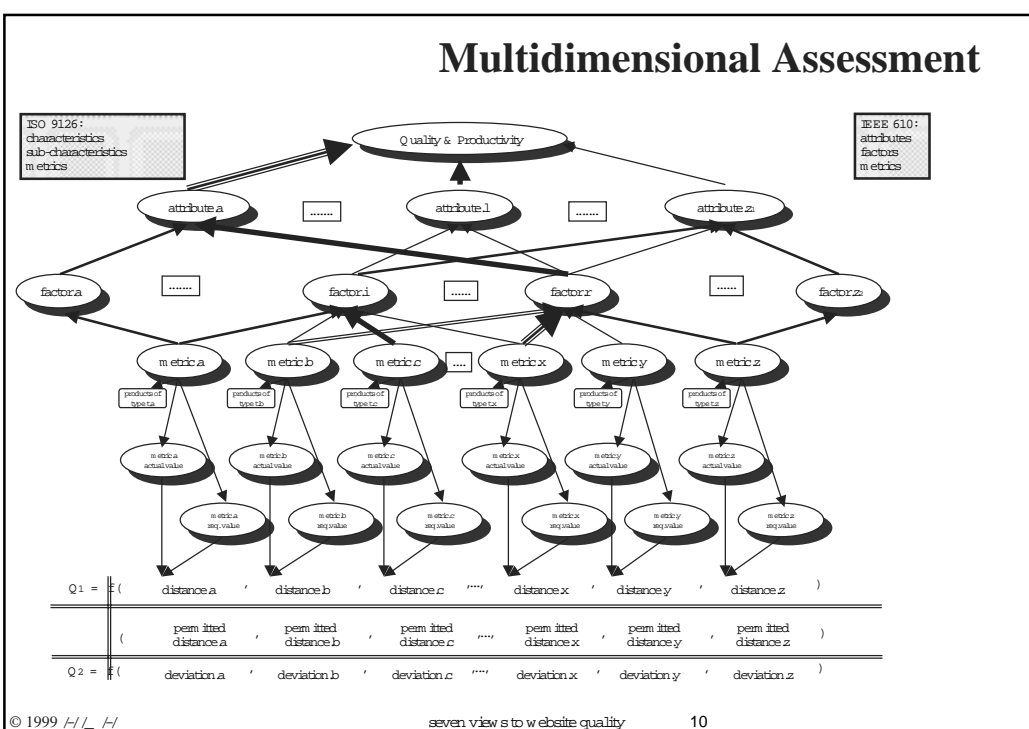


THOUSANDS OF METRICS

Minimal Metrics Set



Multidimensional Assessment



Website Components

Website := Set of linked Digital Objects + Service

Digital Object := Hypertext denoting:

Executable Code, Audio Objects,
Picture Objects, Video Objects.

Digital Object is a highly structured data structure or data base

Service := Result obtained when interpreting

Executable Code

and/or applying tools to digital objects

Service is like interpreter output plus transfer according a protocol

A Digital Object

*object := <object-id, object-location, object-author,
object-neighborhood, object-content, object-extra>*

which might be decomposed into

<object-id> := <object-number>, <object-name>

<object-location> := a URL

<object-author> := web-site-manager

<object-neighborhood> :=

set of URLs or URNs of the objects semantically closed to the actual object

<object-content > := set of relevant descriptors for the particular object V

<object-image-content > V

<object-sound-content > V

<object-text-content >

The base scheme can be made extensible and enhanceable by introducing typed components, thus we get:

object := <object-type>

< object-id-type: object-id> ,

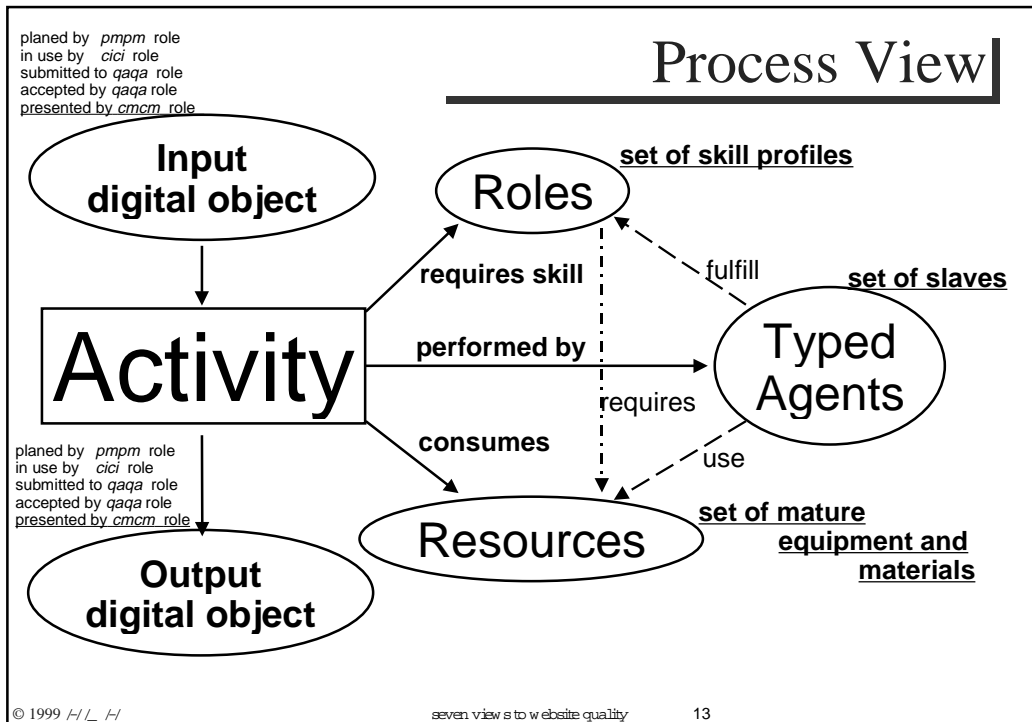
object-location-type: object-location> ,

object-author-type: object-author> ,

object-neighborhood-type: object-neighborhood> ,

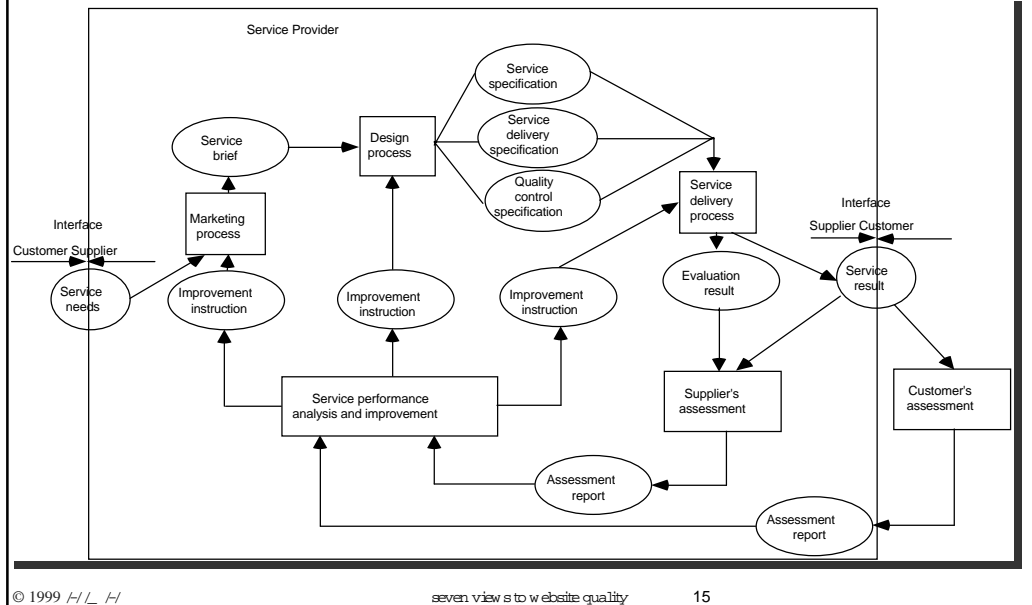
object-content-type: object-content> ,

object-extra -type: object-extra >



- ## Process characteristics
- † Understandability
 - Is the process defined and understandable?
 - † Visibility
 - Is the process progress externally visible?
 - † Supportability
 - Can the process be supported by tools?
 - † Acceptability
 - Is the process acceptable to those involved in it?
 - † Reliability
 - Are process errors discovered before they result in product errors?
 - † Robustness
 - Can the process continue in spite of unexpected problems?
 - † Maintainability
 - Can the process evolve to meet changing organisational needs?
 - † Rapidity
 - How fast can the system be produced?
- © 1999 H.L.H. seven views to website quality 14

Quality of Service Evaluation



© 1999 H.L.H.

seven views to website quality

15

As usual the Questions are

1. Process " *What has to be done?* "

What is to be determined here is which activities have to be carried out, which results have to be produced.

2. Methods " *How is this to be achieved?* "

What is to be determined here is with which methods the activities laid down for the first level are to be carried out and which presentation means are to be used in the results.

3. Tools " *By what is something to be done?* "

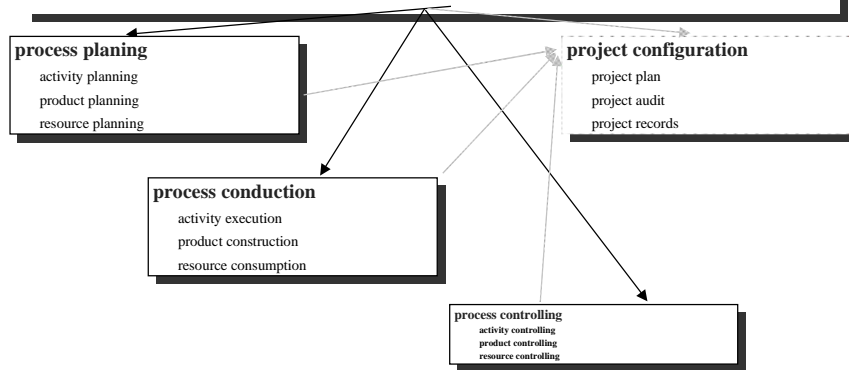
What is determined here is which functional characteristics the tools have to have which are to be used in the software development process. On all three levels, the regulations are structured according to activity areas.

© 1999 H.L.H.

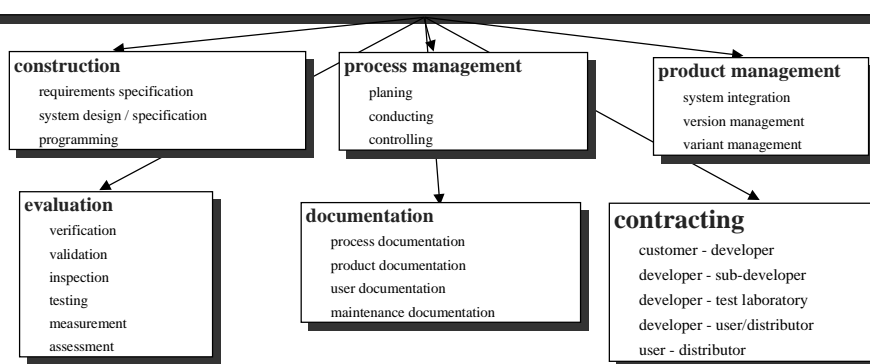
seven views to website quality

16

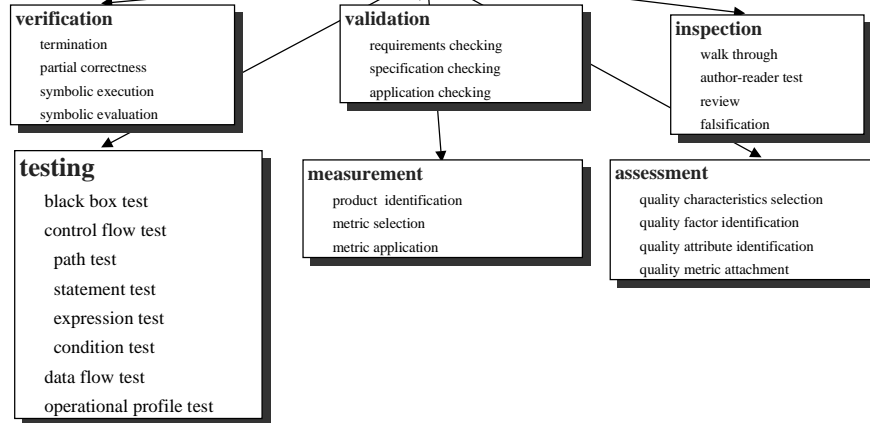
Software Process Elements



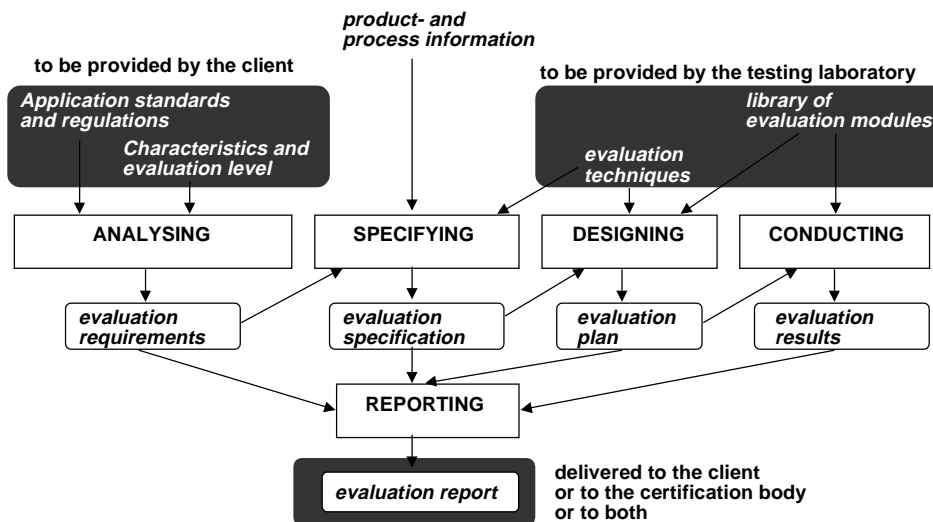
Components of a Method



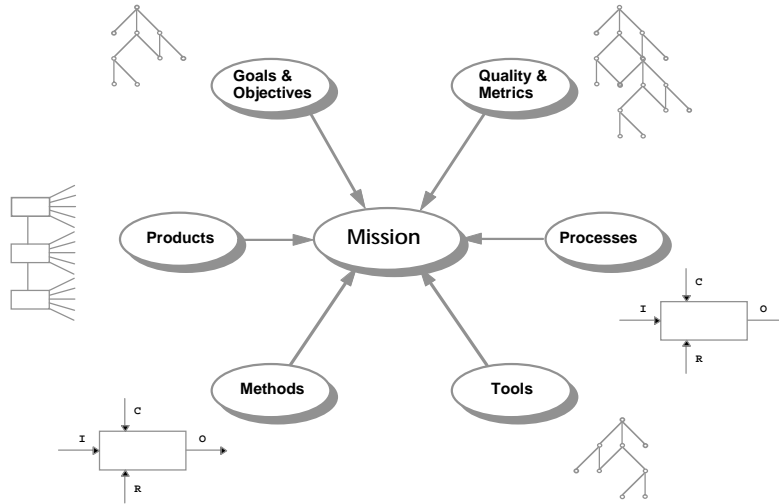
Components of a Software Evaluation Method



Evaluation Procedure



Data Model for Y7



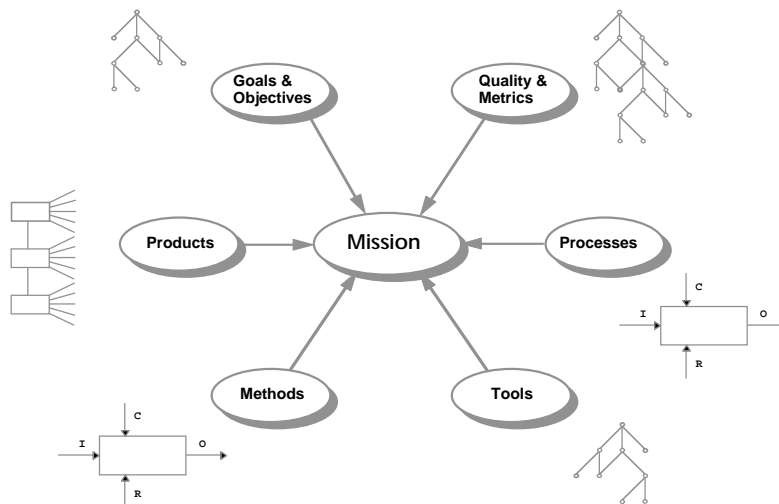
© 1999 H.L.H

seven views to website quality

21

Data Model for Y7

Data Model for Y7



© 1999 H.L.H

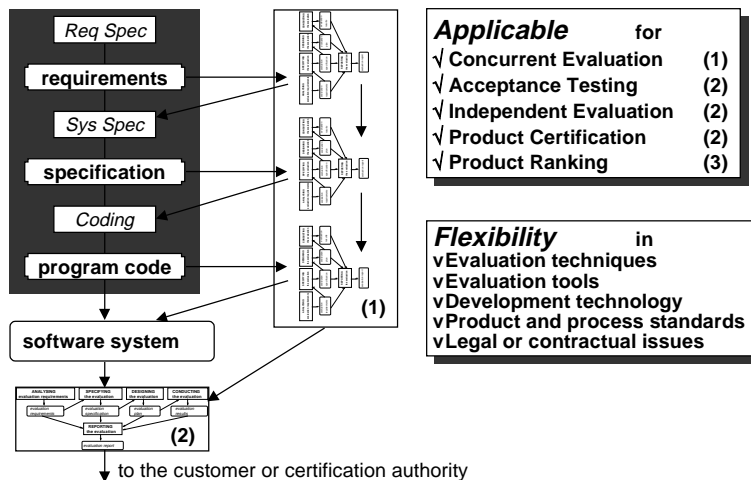
seven views to website quality

22

Aspects of Evaluation



Application of the Method



RADIUM – Applying RAD to innovative ERP/ E-commerce projects

C. Baresi, G. Bazzana, G. Rumi
Onion S.p.A., Italy

Abstract

ERP systems are experiencing a tremendous growth owing to new requirements imposed by the evolution of the market and its new paradigms. Introduction of such systems is a big effort, both from an organisational and technical point of view. On the other side Internet and its new technologies are growing faster and faster and give to companies the opportunity to reach new markets. The integration of ERP systems and Internet technologies is the only way to satisfy new market requirements.

This paper presents the experiences matured in the introduction of SAP R/3 following an "on spec, on time, on budget" strategy. The methodological approach combines the adoption of ASAP (Accelerated SAP, a methodology specifically intended for reducing delivery time) with innovative approaches like management on Intranet of the project key information. The approach has been successfully applied (within the scope of the RADIUM project) both to the introduction of classic ERP features as well as to the integration of ERP features with WWW environment.

Keywords

Rapid Application Development (RAD), Enterprise Resource Planning (ERP), SAP, Accelerated SAP (ASAP), E-Commerce

1. Introduction

As we approach the year 2000, the development, use and management of information systems in enterprises will continue to undergo revolutionary changes as businesses continue to respond to aggressive competition and the need to achieve increasing levels of customer satisfaction. Moreover, companies are discovering that old solutions do not work with new challenges. The business parameters are changed, and so the risks and payoffs.

The Internet and its various technologies have established themselves as a new communication medium. Marketing via the Internet is already playing an increasingly important role in electronic commerce, and further substantial increases in sales volumes are expected. The increased number of people on the web has given businesses a new way to reach customers and enhance profits.

The real power of the Internet today is to provide universal access to information. For businesses, the web can empower customers, partners and employees to access information

directly in a new, self-service paradigm, while reducing costs in business operations and IT infrastructure.

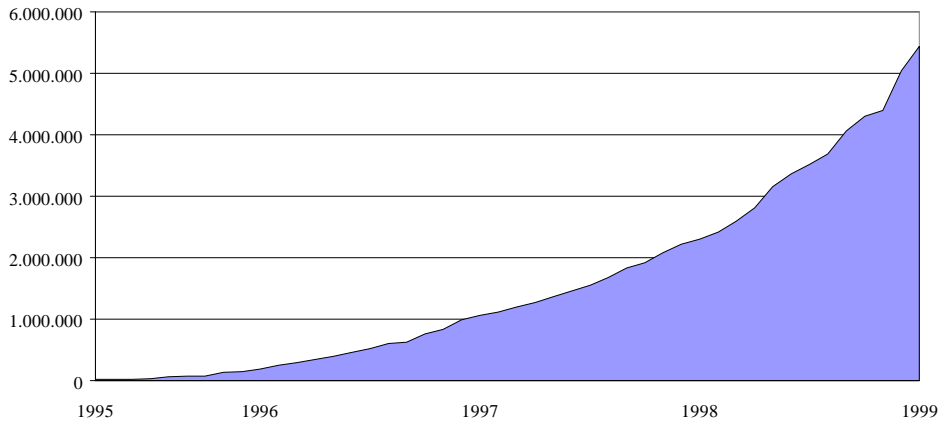


Fig. 1: Actual WWW Growth (number of servers)

Today, electronic commerce represents an unprecedented opportunity for companies to reach new markets, reduce selling and marketing costs, and enhance their relationships with customers and suppliers. Electronic commerce has opened a new universe for consumers and organisations, and it demands new management approaches. Electronic commerce is also very important in internal organisational functioning, as evidenced by the rapid proliferation of Intranets.

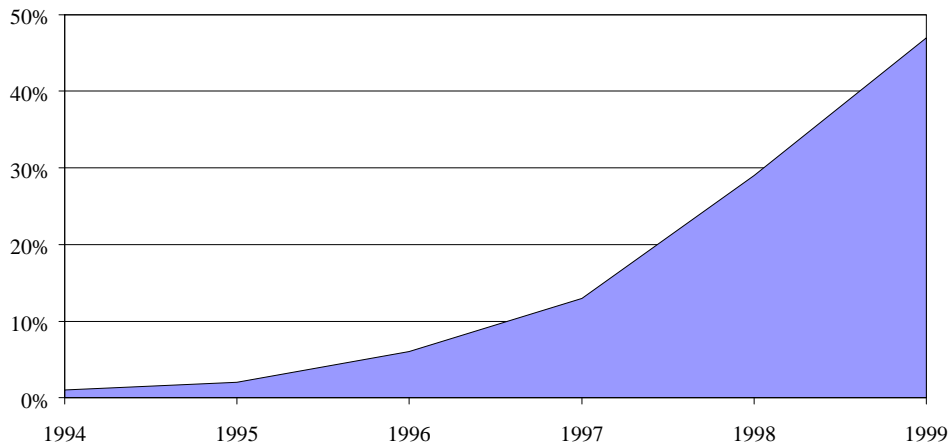


Fig. 2: E-commerce Growth (% of companies expecting benefits from E-commerce)

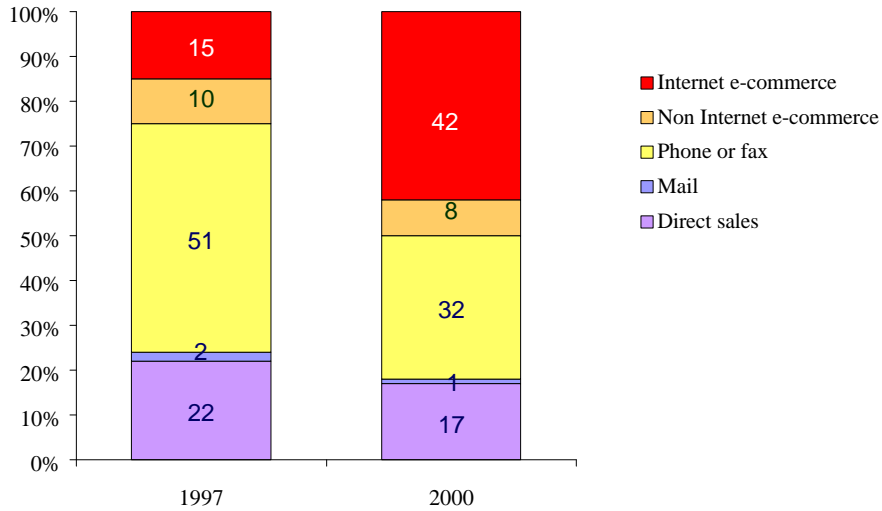


Fig. 3: Increased weight of the E-commerce sales channel

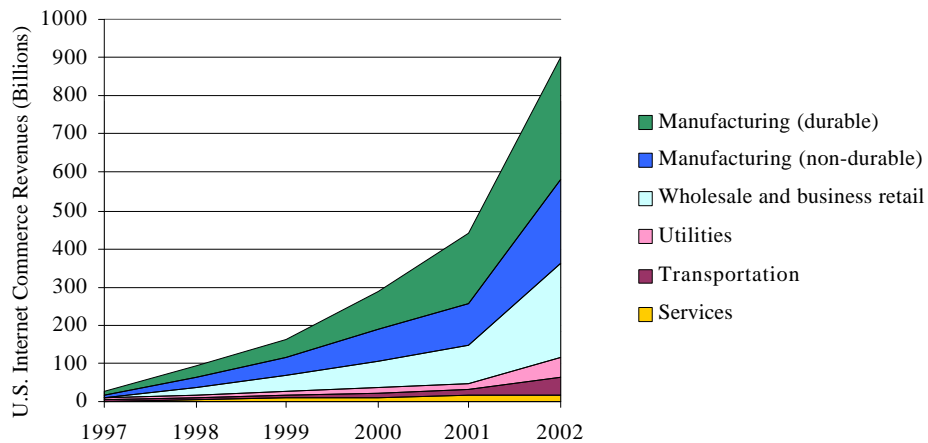


Fig. 4: Penetration of e-commerce by domain

The emergence of client/server computing environments, along with the growing demand for process reengineering to address these rapidly changing business requirements, has created a growing demand for ERP System. Given the tremendous pressure on business, the modernisation of their information systems to meet these challenges will only succeed if organisations effectively develop, manage and use an ERP System.

Today companies use the Internet for marketing purposes. The only way to satisfy new market requirements is to fully integrate ERP software with Internet technologies.

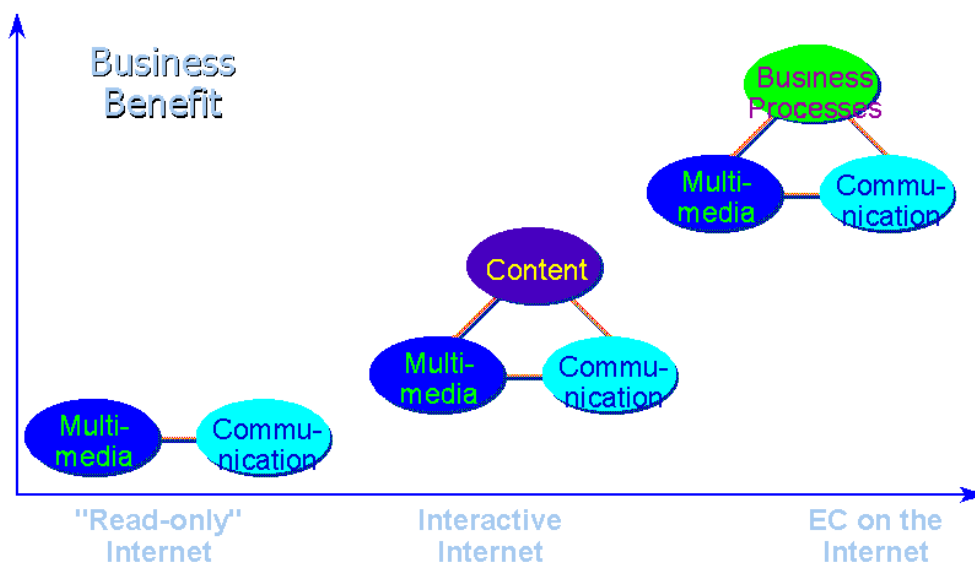


Fig. 5: SAP R/3 roadmap for business on the Internet

In order to make the introduction of ERP systems and their integration with E-commerce “on spec, on time, on budget”, a new methodology must be approached. The integration between ERP and the Internet requires the application of new development paradigm to obtain fast and reliable deployment. An example of such an approach and the related methodology will be given in the following paragraphs.

2. The RADIUM Project

The RADIUM (Rapid Application Development Improvement Using Multimedia) Project had the main goal to improve the software development capabilities of ONION, an Italian organisation offering ICT services (for more details see the Web at URL: <http://net.onion.it/>). While supplying its services, ONION has more and more to face with the development of innovative projects, where the complexity is related to the integration of different media technologies, and integration projects, where the complexity is related to the merging of different systems and different technologies.

The RADIUM Project has constituted an important component of the Software Process Improvement Program that ONION is undergoing. The Process Improvement started in mid 1995 and was based on an initial self-assessment aimed at singling out strong and weak process areas. Starting from the assessment, the Improvement Plan has been derived, and is now driving all ONION software process improvement actions.

The RADIUM Project has represented the evolution of the ONION commitment to the improvement of software development capabilities. In particular, it concentrated on the selection and widespread adoption of methods and tools for the Rapid Application Development of innovative projects, tightened by short development times. Focus of the whole activity was put on the initial phases of the development life cycle, in particular requirements management. This was done in order to better the definition and structuring of requirements and fast prototyping, as well as to produce easily modifiable incremental prototypes in a fast way, with a satisfactory degree of quality.

The approach was driven by both methodological and technical needs, specifically related to the definition of the reference development model and to the selection and adoption of the most appropriate tooling for supporting the activities.

Considerable effort was put into training and dissemination activities (both internal and external) in order to raise the “company quality culture” and to perform appropriate technology transfer.

Moreover experiences were packaged in the form of new/ enhanced standard operating procedures within the ISO 9001 compliant Quality Management System.

Going just a bit into details, the RADIUM project were structured into the following main activities:

- technology survey;
- technical set-up of the software factory;
- definition of RAD life cycle;
- definition of Requirement Capture methods;
- definition of prototyping methods;
- application of the defined methods/ tools to the first baseline project;
- application of the defined methods/ tools to the second baseline project;
- measurement of results;
- deployment decisions;
- alignment of the standard operating procedures.

In the remainder of this paper we describe the experiences learnt from the second baseline project, which focused on methods and tools to shorten delivery time of SAP ERP Projects, in particular when delivering E-commerce features.

3. Improving timeliness in SAP / E-Commerce deployment

3.1. The baseline project

The SAP System (<http://www.sap-ag.de/>) constitutes one of the most popular ERP solutions for the integrated management of the company business. The SAP R/3 System, can be distinguished by the following most relevant characteristics:

- *it is an integrated and comprehensive system*, as it addresses all the company business sectors such as sales and distribution, financial accounting, warehouse management, production planning and human resources;
- *it is an open system*, for on the first hand it is possible to adopt only the really needed modules and on the second hand it runs on the hardware platforms of leading international vendors and on the third hand it allows interoperability with third party solutions;
- *it is always updated* so to be able to use new technologies, as showed by the recent integration to Internet and e-commerce;
- *it is international*, since it covers all the legal and financial issues which are peculiar in the different world countries, so that multinational companies can use it all over the world without any additional integration effort.

The baseline project was constituted by the installation of SAP core modules (FI; CO; MM, PP; SD; WM; QM) to an international manufacturing group for an effort of over 10 person-years along 15 months.

Also E-commerce features were put to trial.

SAP electronic commerce solutions are envisaged to provide:

- links between Management Information Systems and WWW;
- access to Management Information Systems via WWW interface;
- integration between web-based application, based on transaction-oriented business processes, and Management Information Systems;
- possibility to set-up simple, cross-platform applications on top of a simple-to-manage and more centralised IT infrastructure.

The RAD approach embedded in the ASAP methodology was applied to the introduction of SAP R/3 and e-commerce, which are generally well known as time consuming and long lasting projects, to shorten the development time frame.

3.2. Applying ASAP to minimise delivery time

ASAP (Accelerated SAP) is SAP methodology to streamline R/3 projects. ASAP envisages to optimise time, quality and resources. It focuses on the co-ordination of the following elements:

- the *Roadmap*, a detailed project plan with detailed descriptions about what, why and how certain activities are performed; the plan describes all the activities in an implementation and includes details related to technical aspects;
- a specific set of *Tools* to support project management and business process re-engineering activities (technical guidebooks and checklists are included); tools are in the form of files, templates and collection of experiences from other projects.

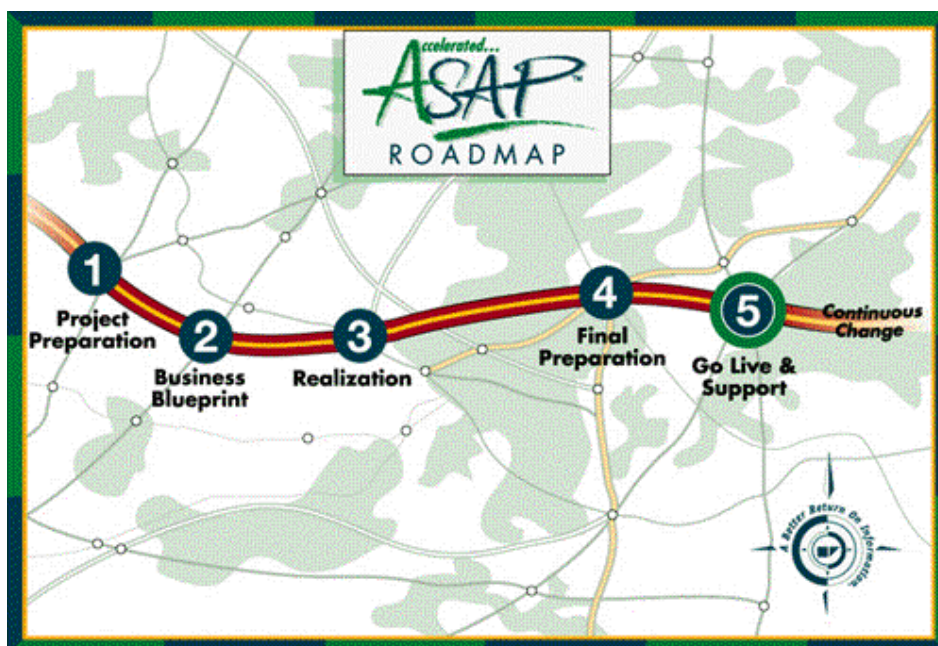


Fig. 6: The ASAP Roadmap

Wherever possible, the ASAP provides examples, checklists, or templates as samples for e.g. a cut-over plan. They are used as a starting point to avoid "reinventing the wheel". ASAP calls these things "Accelerators".

Another very useful ASAP tool is the *Question and Answer database*. It is a repository of all questions that are needed to define business requirements and to develop the business solutions in terms of the R/3 Reference Model and R/3 System. This includes business processes, technical, organisational, and configuration questions and answers that are the source for creating the Business Blueprint. It gives the possibility also to collect company responses and to automatically produce documentation starting from the database.

Further to guaranteeing a close control over planned activities and progress status, the ASAP methodology makes it possible to significantly shorten the implementation and customisation of a R/3 SAP system. Moreover it helps in defining QMS rules related to the introduction of Management Information Systems and it allows to follow these rules during the project.

In short, ASAP helps in optimising time, quality and efficient use of resources, focusing on the co-ordination of all activities to ensure the successful completion of a SAP R/3 project. The result of our experiences is that the application of ASAP to minimise delivery time is worth when considering the result of the following topics which are clearly addressed by the methodology:

1. *Goal-oriented project management*: this aspect is coupled with a detailed project plan that predefines milestones and quality checks. In particular, ASAP provides strategies and recommendations that have to be followed and helps in evaluating critical needs vs. "nice-to-have" and how to implement a process-oriented implementation.
2. *Assured quality and know-how transfer*: ASAP has predefined quality checks at the end of each phase that offer the opportunity to perform quality reviews that evaluate progress and risk factors step by step. The output of these checks are detailed progress reports and formal acceptance of each phase by the steering committee.
3. *Efficient use of resources*: ASAP takes this into account and gives recommendations about the needed background, experience and other skills that the project team members need to have.
4. *Consistent documentation*: ASAP delivers a documentation concept for all types of documentation that are typically created during an implementation (project quality plan, project plan, meeting minutes, progress reports, technical documentation, issue and change requests, etc.); in this respect it is quite easy to work with pre-defined documents and templates. It is also possible to adapt these documents and templates to be ISO9001 consistent.
5. *Insight into the implementation process*: with the ASAP Roadmap, it is very easy to follow the implementation process. In particular, it is quite easy to understand at each time where we are, what is planned and which are the deliverables of each phase. This fact also helps not to forget anything and to co-ordinate all the links between business and technical tasks.
6. *Emphasis on early stages in the project*: ASAP addresses a lot of activities usually underestimated or started too late such as data conversions, interfaces, and authorisations.

The application of the ASAP methodology force the Project Manager to define a very detailed Quality Plan for the project. It helps in the definition of all the activities needed to successfully complete the project and consequently in an accurate definition of the budget. It is an essential tool to ensure that your project is "on spec, on time, on budget".

All the tools at Project Manager's disposal are very useful to help key users in analysing processes and in defining the final solution, without forgetting any part of the system and focusing on real needs.

Moreover, ensuring a very strict check on the project progress helps in preventing problems and in selecting the best solutions.

3.3. Intranet

Another very important role within the development activities is played by the diffusion and sharing of information through both the company staff and the development team. To make it as free and easy as possible we experienced that the management of all the key information via Intranet constitutes a winning aspect. In fact, a specific area within the company Intranet has been set-up to manage all the relevant documentation such as:

- Project overview;
- Planning and tracking;
- Project rules;
- Meeting minutes;
- Technical documentation;
- Issues and change requests.

Appropriate access levels have also been set up in order to guarantee security in respect to the stored information.

3.4. Applying RAD methodology to E-Commerce trial

Web retailing on the Internet plays an important role in electronic commerce. With the use of the Internet the standard supply chain is extended with customers and suppliers accessing the Information System from outside.

There are three different classification of web-based applications:

- Business-to-Consumer (Internet): the customer uses a Web browser to access the vendor's system to review a product catalogue, place an order, or inquire about a product or service;
- Business-to-Business (Extranet): integrated business systems can co-operate with each other; information such as order numbers, customers, and invoices is exchanged;
- Intranet: the Internet is used for communication within a single enterprise.

SAP has two different approaches to the development of Internet applications:

- **Inside-out:** information is exported from the Information System to the web; Internet Applications Components (IACs) and SAP Internet Transaction Server (SAP ITS) are used. IACs are ready-to-use solutions able to access information stored in the SAP system via BAPIs (Business Application Programming Interfaces), assuring complete compatibility with future releases. Moreover, the IACs can easily be tailored to your specific requirements by designing the Web pages used in the component accordingly. On the other side, the SAP Internet Transaction Server (running on Windows NT) combines Internet technology with SAP technology, providing consistent information, scalability and security.

- **Outside-in:** information is accessed from an external web-based application. “Ad-hoc” applications are developed using BAPIs to access information on the SAP system. The “standard” Internet technology is used to export information. BAPIs provide a standard, multi-vendor interface for common business processes and objects such as creating customer orders or customer master records. In this way, the application is completely apart from the SAP system and can interact with many other systems, even if different core technologies are used.

In all Web-based applications, security is one of the most important theme. Even more in applications publishing on the Internet information from the Information System and in application related to e-commerce.

The needed level of security is assured by four different group of features:

- *ITS architecture:* the integration of SAP with E-commerce is provided through a multi-tier architecture, as detailed in the figure below.

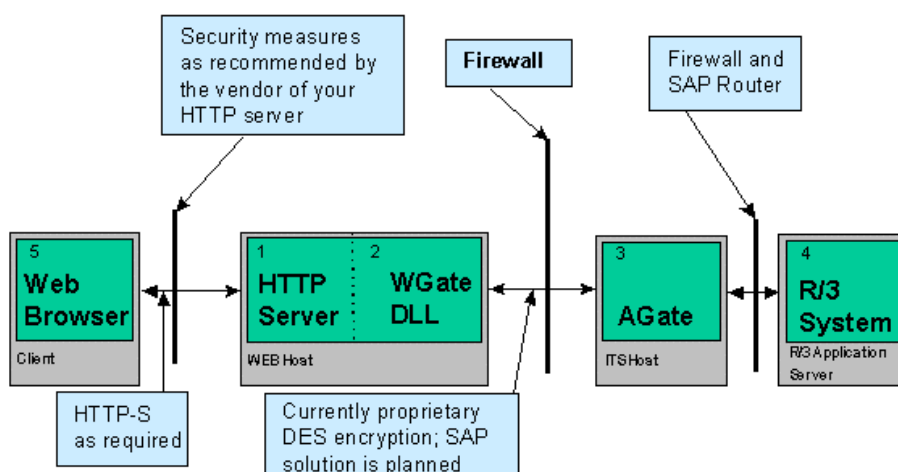


Fig. 7: Multi-tier architecture

To tighten the security of your ITS installation you can add firewalls both between the ITS server and the R/3 Application server and between the Web host and the ITS host.

- *NT security features:* some ITS information, such as passwords and files needed for ITS execution, must be safeguarded against unauthorised access; it is recommended that you restrict access to this information as much as possible, using Windows NT security.
- *R/3 security features:* the ITS needs an R/3 user account to access the R/3 System. Depending on the needed level of security, you can decide to create a **global** R/3 user account for all services, or a **specific** R/3 user account for each individual service, or **no static** R/3 user account (in this case, the user is prompted for the user name and password in a logon procedure each time the service is started). Static R/3 user names and passwords are stored in the service files. All passwords are encrypted. If you use the secure-installation option, all service files are additionally protected by the NT security system.
- *ITS security features:* you can specify that the data passed between ITS host and web host be encrypted; moreover ITS has appropriate internal mechanisms designed to prevent unauthorised third party to connect to an existing R/3 session.

4. Conclusions

The application of the ASAP methodology has made it possible to achieve significant results:

- first of all it has been essential to keep to schedule the implementation and customizing of our R/3 SAP system; it has to be underlined that the project was expected to require 40% longer schedule in accordance with SAP estimation rules based on historical SAP projects developed in conventional ways;
- the documentation produced using the ASAP instructions is completely consistent with the ISO9001 standards; as a confirmation of that, the project has been inspected by a certification body, during an ISO9001 surveillance visit and no anomaly has been detected;
- last but not least, ONION has won the ASAP Award as one of the best SAP Project in Italy managed using the ASAP methodology

From a technical point of view, the application of the ASAP methodology allowed the project to reach the following goals:

- proper and clear definition of project objectives;
- involvement of management and key users in all phases of the project;
- strict control on project progress and timely reporting to project management;
- achievement of deadlines;
- reduction in delivery time.

The SAP e-commerce solutions offer different benefits:

- you can take advantage of the SAP International capabilities with very little effort;
- you do not need to redefine sales processes due to the complete integration with the R/3 processes;
- there is no need to duplicate data since there is only one database;
- due to the ready-to-use solutions, it is very simple and fast to produce a prototype;
- SAP guarantees compatibility with R/3 future releases.

5. Acknowledgements

Our acknowledgement goes to the members of the ONION ERP development staff contributing to the project and in particular to Mrs. E. Zanelli, Mrs. S. Rodella, Mr. E. Gares, Mr. S. Peli and Mr. G. Zontini, who experienced and intensively co-operated in applying the new approach.

We have to thank SAP Italia for the initial support in acquiring the basis for the ASAP methodology and the SAP development team in Walldorf who has always been available for support.

Moreover, we have to thank the European Commission for the financial support given to the RADIUM Project (Number 23842), run under the ESSI Initiative as part of ESPRIT Framework IV Programme. We are especially indebted to the Project Officer, Mrs. Bogliolo, for her support.

6. Contact Point

Gianni Rumi

ONION S.p.A.

Via L. Gussalli, 9 – 25131 Brescia, Italy

Tel. +39.030.3581510

Fax. +39.030.3581525

E.mail: gr@onion.it



Internet/Intranet/Extranet for Business Process Re-Engineering

QWE 99



RADIUM

Applying RAD to innovative ERP/E-commerce projects

G. Bazzana, C. Baresi, G. Rumi

Bruxelles - Nov 5, 1999

ONION S.p.A. - Profile



- Based in Italy
- Business Areas:
Communications/
Technologies,
Consulting/ ERP
- Continuous Process
Improvement since 1996
- ISO 9001 registered for
all provided services



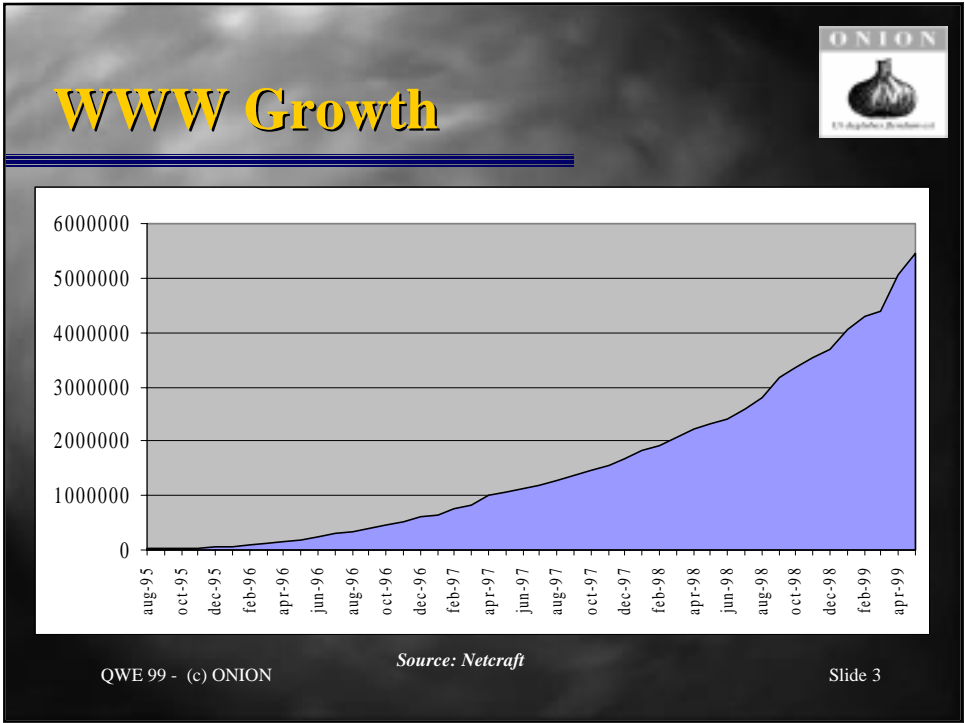
<http://www.onion.it/>

QWE 99 - (c) ONION

Slide 2

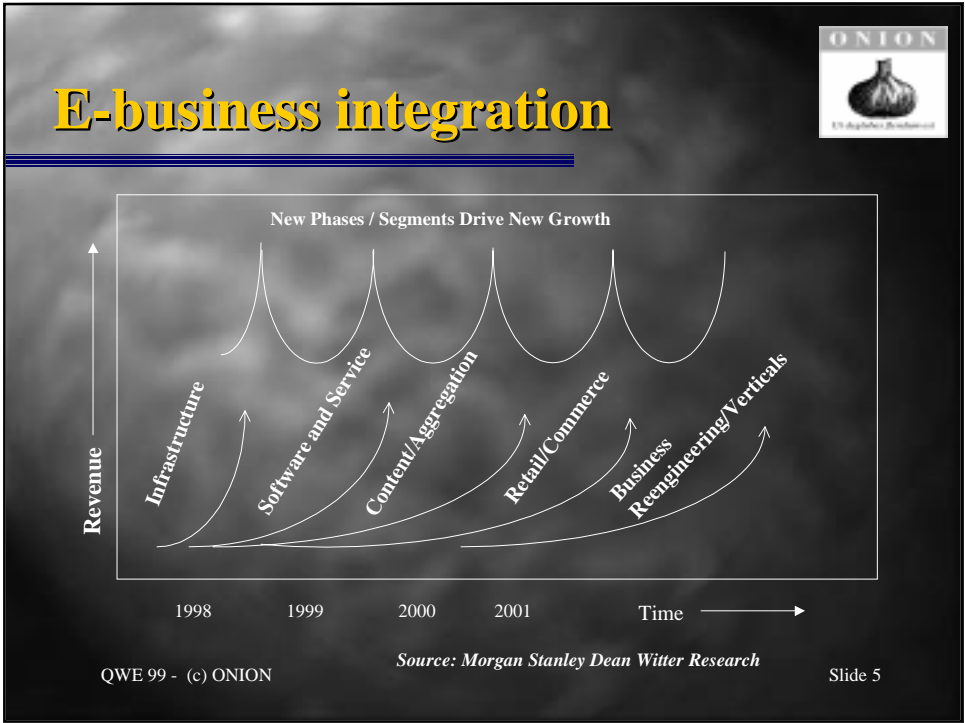


Internet/Intranet/Extranet for Business Process Re-Engineering





Internet/Intranet/Extranet for Business Process Re-Engineering





Internet/Intranet/Extranet for Business Process Re-Engineering

The Challenges



- 1 - Process Re-engineering
- 2 - Time to delivery
- 3 - Quality of service



QWE 99 - (c) ONION

Slide 7

RADIUM Project goals

- Improvement of software integration capabilities for innovative development
- Reduction in time to market while keeping QoS
- Application of RAD concepts to ERP projects
- Definition of most suitable techniques for integration between ERP and E-commerce

QWE 99 - (c) ONION

Slide 8



Internet/Intranet/Extranet for Business Process Re-Engineering

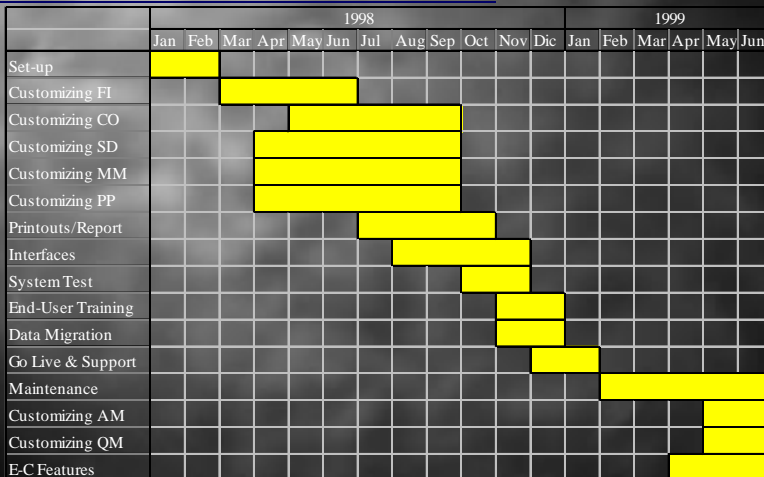
Technical activities

- technology survey
- technical set-up of the software factory
- definition of RAD life cycle
- definition of Requirement Capture methods
- definition of prototyping methods
- application to the 1st baseline project
- application to the 2nd baseline project
- measurement of results
- deployment decision
- alignment of the standard operating procedures

QWE 99 - (c) ONION

Slide 9

The Baseline Project



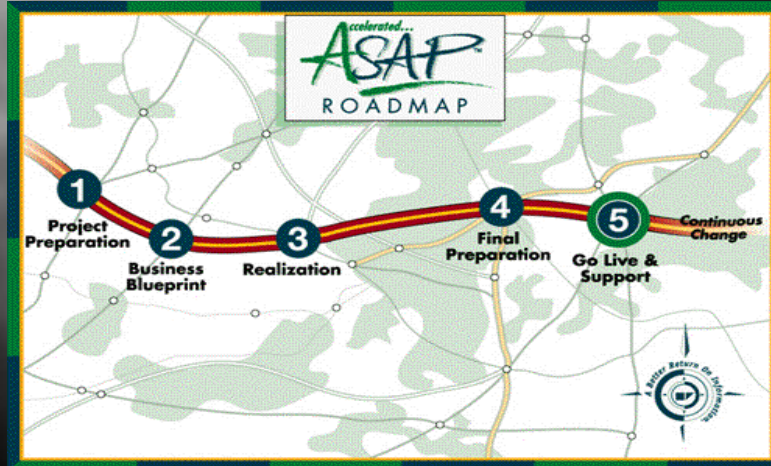
QWE 99 - (c) ONION

Slide 10



Internet/Intranet/Extranet for Business Process Re-Engineering

The ASAP Methodology



QWE 99 - (c) ONION

Slide 11

Process integration

The typical top-down approach of ERP projects (“as-is” --> “to be” --> BPR --> bla, bla) has been contaminated with RAD approach

Round trip engineering:
analyze a little, customize a little, test a little

Strong communication means across project team (> 20 staff members)

QWE 99 - (c) ONION

Slide 12



Internet/Intranet/Extranet for Business Process Re-Engineering

The Intranet as a support

SAP Project

The XXX - SAP Project aims at the introduction of the [SAP R/3 ERP](#) (DISTRIBUTED BY SAP ITALIA) System within the XXX Group. It is jointly managed by [XXX](#), [Onion](#), and [SIC](#) (SAP Italia Consulting).

Look at the project information stored in this WWW

- [Activities in 1998](#)
- [Activities in 1999](#)
- [Technical Environment](#)
- [Hot links](#)
- [ASAP Award](#)
- [What's New](#) - last updated on April 01, 1999

[Get Annual Report](#) - Several PDF files managed in PDF format. If you do not have the Acrobat Reader, click [here](#) to download it.

If you have any comment or suggestion, please share an email to the [webmaster](#).

Web made by [Onion Communications - Technologies - Consulting](#) © 1999

QWE 99 - (c) ONION Slide 13

Technical Results

- .Goal-oriented project management*
- .Assured quality and know-how transfer*
- .Efficient use of resources*
- .Consistent documentation*
- .Insight into the implementation process*
- .Emphasis on early stages in the project*

Major saving of calendar time while keeping expected quality

QWE 99 - (c) ONION Slide 14



Internet/Intranet/Extranet for Business Process Re-Engineering

Management Results

Reduction of schedule
from 15 to 9 months

Full consistency with ISO 9001

The baseline project won
the “ASAP Award”

QWE 99 - (c) ONION

Slide 15

Conclusions



QWE 99 - (c) ONION

- Experiences have been matured in the context of the RADIUM Project under the auspices of EC DG XIII
- Deployment is on-going towards all projects
- Ultimate goal: “on spec, on time, on budget”



Slide 16



Taking Care of E-Business...



Elie Kanaan
Marketing Europe

www.merco-llc.com



Product Information

**At the wake of the 3rd millenium :
1 concern, 1 trend and 1 opportunity**

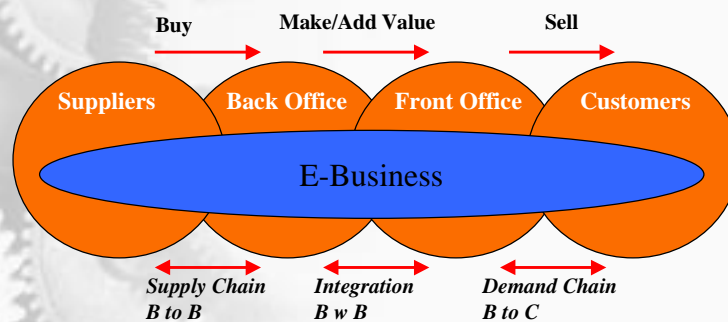
- Year 2000
- Internet becoming the worldwide communication/commerce backbone
- IT as a value delivered to customers thru E-Business applications

Topics

- **[E-Business] Application Testing: Do or Die !**
- **Product Update**
 - Testing : TestSuite Enterprise 6.0 and Astra
 - Application Performance Management : Topaz
- **E-Business EcoSystem**
- **Conclusion**

What is E-Business?

E-Business = Transforming key business processes thru the use of Internet technologies

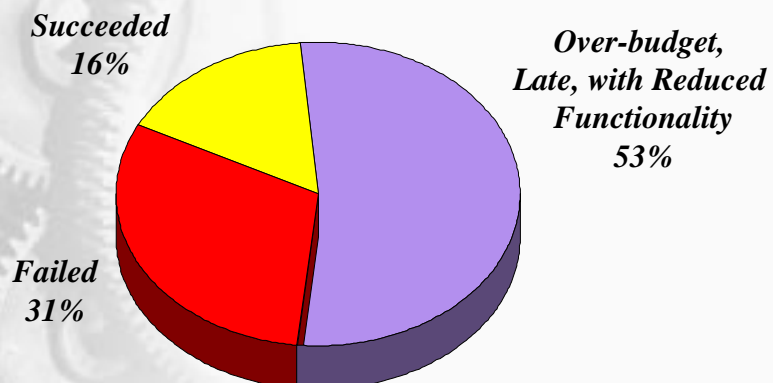


Source: Yankee Group

E-Business = Big Business ?

- Political Risks
 - Executive Sponsoring & Commitment
- Business Risks
 - Business & Selling Model
- Organisational Risks
 - Organisational Structure
- Technical Risks
 - E-Business Systems and Applications

Technical Risks: How Big is the Problem?



The Standish Group Study of Business-Critical Projects



Technical Risks: How Big is the Impact?

| Web Site | Daily Internet commerce revenue as of 1/15/99 | Lost revenue per hour of downtime as of 1/15/99* |
|--|---|--|
| www.techdata.com | \$1,000,000 | \$8,280 |
| www.amazon.com | \$2,700,000 | \$22,500 |
| www.dell.com | \$10,000,000 | \$91,320 |
| www.cisco.com | \$20,000,000 | \$182,640 |
| www.intel.com (partner extranet site only) | \$33,000,000 | \$274,980 |

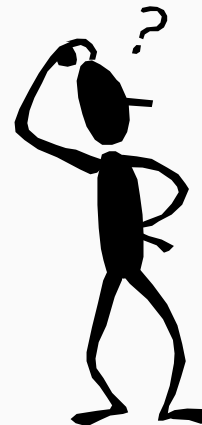
* Lost revenue assumes a \$1 million-per-day site where 20% of transactions are lost during downtime. Costs due to brand erosion and decreased customer satisfaction are not included in these estimates

Forrester Research Inc. Jan. 1999



Why Continuously Test E-Business Applications?

- **If the Application breaks the Business stops**
- E-Business applications are Part of the product sold to customers
 - Poor customer experience = lost customers and business
- Application Performance and Reliability = Business Performance
 - Too Slow, never again
 - Not available, we go elsewhere





“8-Second Rule” Impacts Revenue

- Web page download time is key in determining success and user satisfaction
- Up to \$4.35B annually in e-commerce sales may be lost due to unacceptable download times!

*Zona Research, 1999

Testing helps protect revenue stream!



Non-Stop E-Business is The Driver

- 92% of users chose reliability as most important feature *
 - Application reliability means higher value to consumers
- 46% of users have on at least one occasion been driven to alternative sites because their preferred site failed*
 - Application reliability protects revenue stream

* Jupiter Communications Survey - Date

What is different about the Internet?

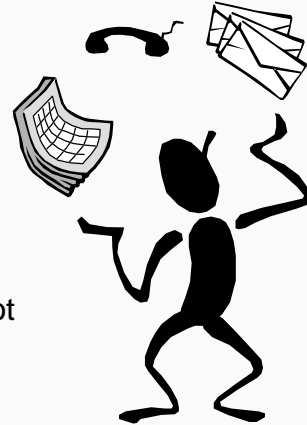
■ Complexity

- Evolving new technologies
- Combinations of technologies

■ Unpredictability of usage

■ Dynamic Content

- Applications change daily, if not more often
- 1-to-1 : customization of user experience



.Com's Experience Barriers to Testing

■ No time

- Can't delay production for lengthy testing cycles

■ No resources

- No testers, no hardware, no budget

■ No testing expertise

- Traditional testing tools are intimidating





Introducing: Astra & TestSuite Enterprise 6.0

- **Astra** - Fast, simple web testing
 - Astra QuickTest, Astra LoadTest, Astra SiteManager
 - ActiveScreen technology simplifies testing
 - Architected from the ground up for web testing
 - First testing tools you can “try & buy” on the Web



- **TestSuite Enterprise 6.0** - an Integrated suite of Enterprise testing tools...
 - TestDirector 6.0 for test management
 - WinRunner 6.0 for functional & regression testing
 - LoadRunner 6.0 for load testing

...ensure higher reliability and a positive end-user experience



Astra





Astra

Fast, simple web testing



Astra QuickTest (HTML Client)

- Verifies that web applications work as expected by mirroring end user behavior to shorten testing cycles



Astra LoadTest (HTTP/S Protocol)

- Validates web site performance by generating load of hundreds of users to identify and pinpoint bottlenecks



Astra SiteManager

- Visual web site management tool



www.astratryandbuy.com First “Try & Buy” Testing Site




- 1 Download Astra from the “Try” site to evaluate
- 2 Come back to web site to “Buy”
- 3 Web-based support



TestSuite Enterprise 6.0



www.merc-it.com


Product Information


WinRunner Supports the Widest Range of Enterprise Environments

Web & Java

- Animated Images
- XML
- JDK 1.1.8 & 1.2.x
- Oracle Jinitiator
- Symantec Café 3.0
- JFC 1.1
- Silverstream

Custom C/S

- VB 4, 5, 6
- PB 4, 5, 6
- Oracle Dev 2000
- Forte 3.0.I.1
- Delphi 2, 3, 4



E-Biz

- IE 4.x, IE 5.x
- NS 4.0 & higher
- DHTML
- AWT from JDK 1.1.7
- JFC 1.0.x
- Symantec Visual Café 2.5
- Oracle Dev 1.6.x (web)

Legacy

- 3270 Emulators
- 5250 Emulators
- vT100 Emulators

ERP

- Baan IVc, V
- SAP R/3 3, 4
- Oracle Apps 10, 10.7, 11
- Peoplesoft 6, 7, 7.5



LoadRunner Supports the Widest Range of Enterprise Environments

Object & Web Protocols

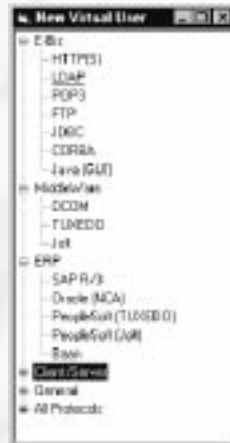
- CORBA
 - Visibroker, Orbix, M3
- DCOM
- JDBC
- LDAP
- POP3

E-Biz

- HTTP
- SSL
- NTLM
- Digital Certificates

ERP

- SAP R/3 3, 4
- Peoplesoft 6, 7, 7.5
- Oracle Apps 10, 10.7, 11
- Baan IVc, V



3-Tier C/S

- Java
- Jolt 1.1
- Tuxedo 6.0-6.4

2-Tier C/S

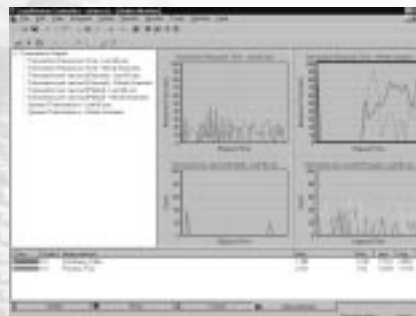
- Oracle 7-8
- MS SQL Svr 6, 6.5, 7
- Sybase 10-11
- Informix 7.1-7.23
- ODBC 2.1- 3.5
- Winsock 1.1

Legacy

- 3270
- 5250
- vT100, vT200, ...
- APPC (AS/400)
- X Windows



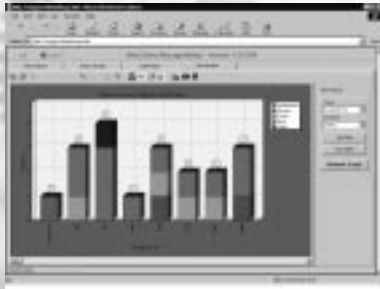
LoadRunner's Integrated Monitors Display Transaction Performance in Real-time



- Identify and isolate system problems as they occur to accelerate problem resolution
 - Monitors performance of transactions, application servers, web servers, TUXEDO servers, network delays, SNMP devices, and databases



TestDirector Provides Access to Defect Management via the Web



A screenshot of the TestDirector web interface showing a table of defect data. The table has multiple columns and rows, with some cells containing text and others containing small icons or status indicators. The interface includes a toolbar at the top and a sidebar on the right.

- Allows anytime, anywhere access through any browser
 - Analyze application readiness status and trends
 - View, insert, and change status of defects



HERCURY INTERACTIVE

Application Performance Management





Application Performance Management Today

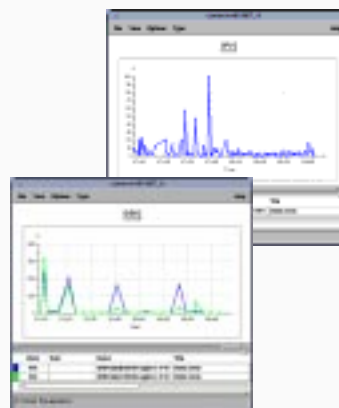
- Performance is not measured at the business process level
 - Various “System” monitors generate confusing data
- Application availability data is not meaningful to end-users
 - Reports are difficult to extrapolate outside of IT
- IT is unable to anticipate application disruptions
 - IT is forced to be reactive to user demands



Current Solutions

Existing tools are net/server centric

- Total requests made per hour/done/outstanding
- Status of each server
- Detailed report on the domain configuration
- Boot and shut down by domain, machine, group and server
- Monitors status of the Process Scheduler
- Monitors status of jobs queued through the process scheduler
- Tracks Process Server statistics



Lots of Data, No End User Perspective

M
MORGAN STANLEY

Product Information

Introducing: "Topaz"

Browser Network Web Server Back End Server

- "Topaz" measures end-user experience
 - Client perspective
 - Proactively measures actual business processes as opposed to system data
 - Provides meaningful data on application performance and availability

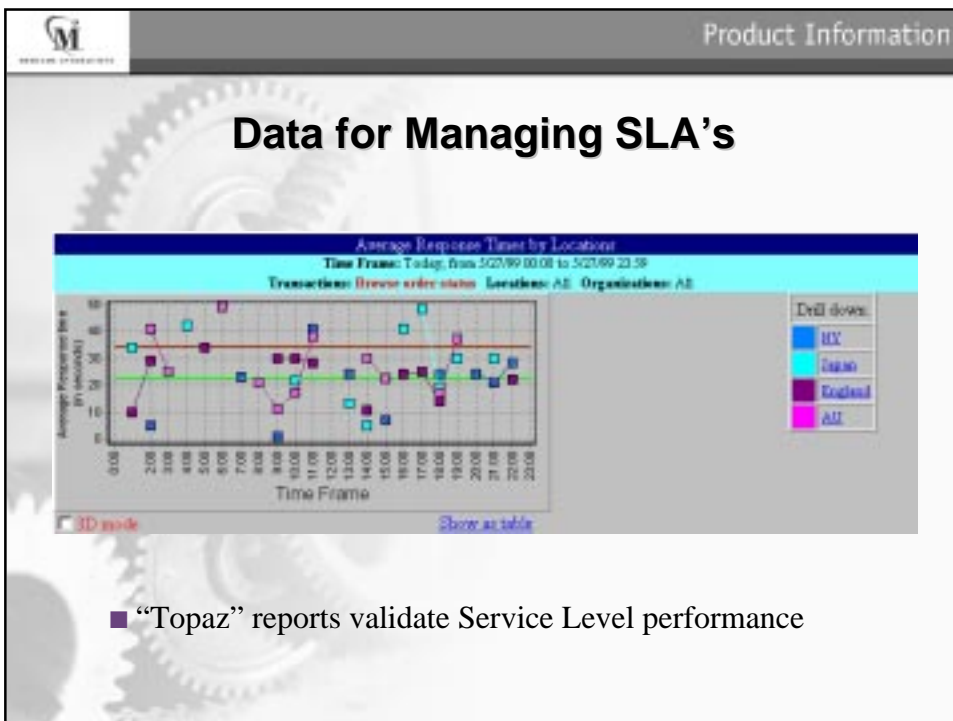
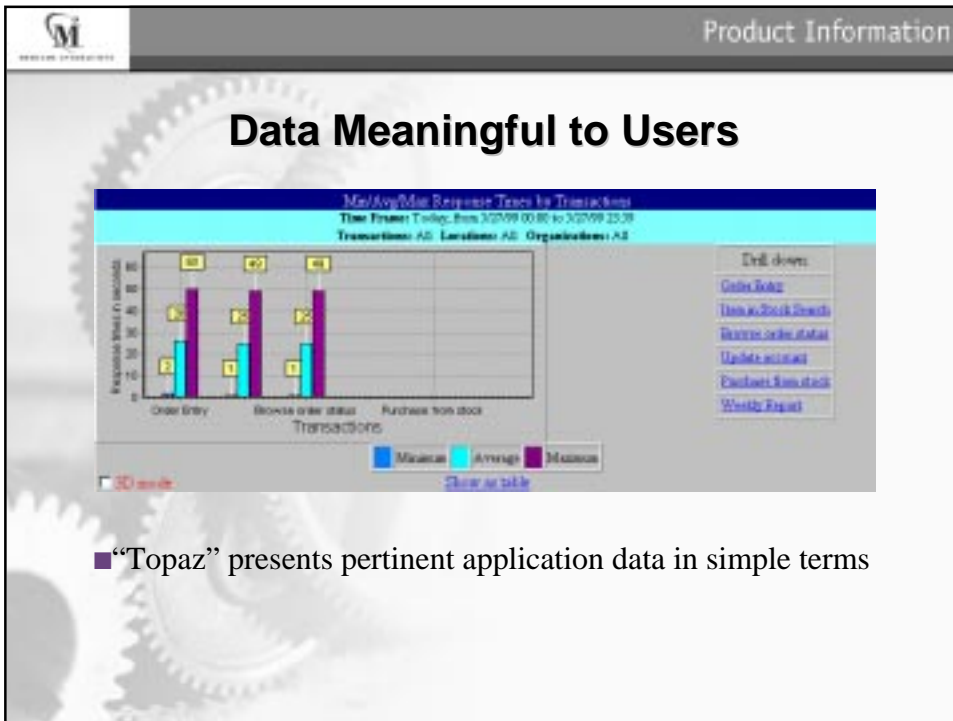
M
MORGAN STANLEY

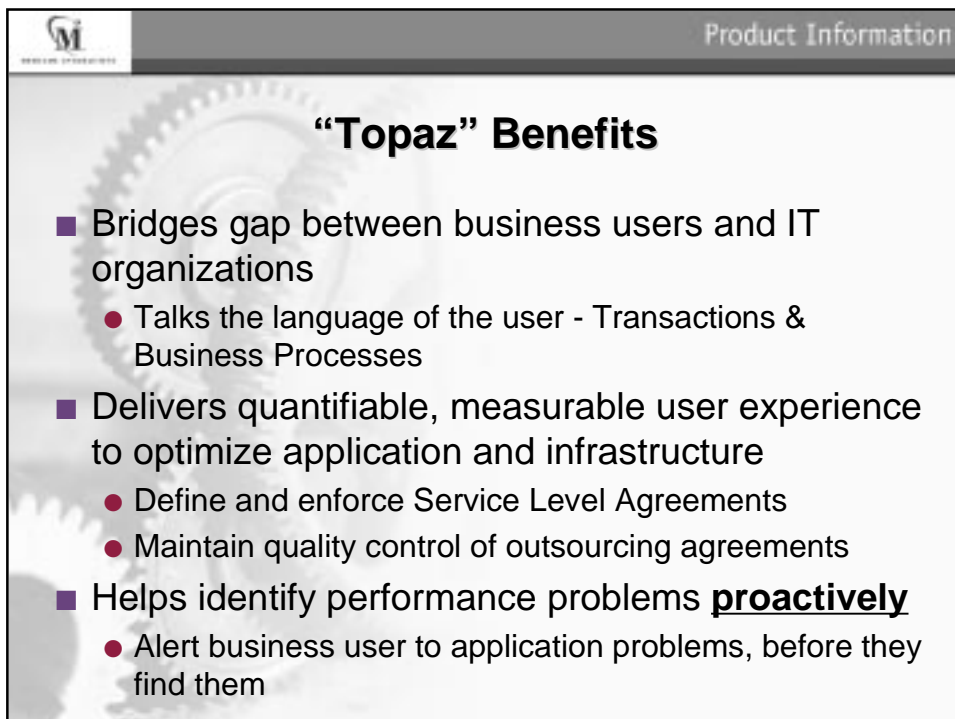
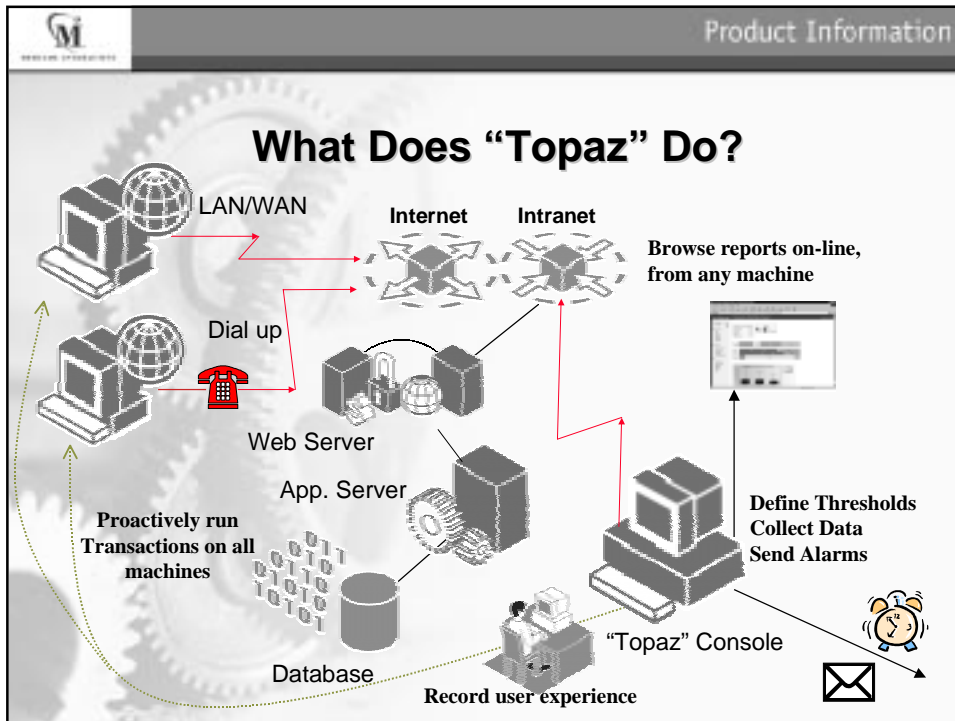
Product Information

Real-time Application Status

| Transaction Health Distribution | | | | | | | | | | | | | | | | | | | | | | | |
|--|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Time Frame: Today, from 02:00:00 to 02:00:21.50 | | | | | | | | | | | | | | | | | | | | | | | |
| Transaction: All Location: All Organization: All | | | | | | | | | | | | | | | | | | | | | | | |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| Order Entry | - | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| Product Search | - | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| Account order status | - | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |

- "Topaz" provides "health" summary of application







M HERCULES INTERACTIVE Product Information

The E-Business EcoSystem

- **New Software Vendors**
 - Application Platforms, Application Servers, Others ...
- **Web System Integrators**
 - Established SIs, New Web-only SIs
- **Internet Service Providers (ISPs)**
 - Exodus, UUNET, Oleana, ...
- **Application Service Providers (ASPs)**
 - Matra Grolier Association, USI, Quest, Corio, ...
- **Packaged Application Vendors**
- **Hardware/Infrastructure Players**

The slide has a background image of interlocking gears. The Mercury Interactive logo and "Product Information" text are in the top left and right corners respectively. The title "The E-Business EcoSystem" is centered at the top of the main content area.

E-Business Alliance Partners

E-BUSINESS ALLIANCE PROGRAM
MERCURY INTERACTIVE

WWW.SUN.COM
Sun microsystems

SYMANTEC.

BROADVISION

VISION SOFTWARE

ARIBA

SilverStream

<allaire>

Bluestone SOFTWARE
Enterprise Interaction Management

CALICO

NCIPHER
SECURING THE FUTURE

GEMSTONE
So you can serve the world

ONESOFT
THE INTERNET COMMERCE COMPANY

Product Information

Select E-business Customers

Fidelity Investments

Gateway
Connect with us

IBM e
an e-business solution

CISCO SYSTEMS
Empowering the Internet Connection

microsoft.com

CITIBANK

jcrow.com

DIGEX

FedEx.

Deutsche Post

AT&T

KLM

Lucent Technologies

America Online

Deutschen Telekom.

Ford

SAB

infoseek

NASDAQ

JPMorgan

priceline.com

American Airlines

bn.com
BANKS & MORE

EXTRADE

Internet Shopping NETWORK

CHRYSLER CORPORATION

MOTOROLA

CATERPILLAR

UUNET

Charles Schwab

Conclusion

- **To avoid the Headlines of Web Disasters, talk to Mercury Interactive**

A collage of news articles and website screenshots illustrating various web outages. The articles include:

- ABC NEWS Tech**: "Another eBay Outage" - New Hours Offered - Cost Company Millions. The article reports that eBay's website was down for over 10 hours on Tuesday, affecting 2.4 million users. It mentions that eBay's website was down for 10 hours on Tuesday, affecting 2.4 million users. The article also mentions that eBay's website was down for 10 hours on Tuesday, affecting 2.4 million users.
- 1 R E D NEWS**: "No Explanation Yet for America's Choice" by Chris DeWitt. The article reports that the website for America's Choice was down for 10 hours on Tuesday, affecting 2.4 million users. It mentions that the website was down for 10 hours on Tuesday, affecting 2.4 million users.
- NEWS.COM**: "Schwab's Net trading site down for an hour" by Michael J. Sauter. The article reports that Schwab's website was down for an hour on Tuesday, affecting 2.4 million users. It mentions that the website was down for an hour on Tuesday, affecting 2.4 million users.
- USA TODAY Tech Report**: "Site outages send customers scurrying" by Doug Levy and Janet Kaminson. The article reports that several major websites went down on Tuesday, causing confusion and concern among users. It mentions that the websites were down for several hours on Tuesday, causing confusion and concern among users.




ESSI Project 27649 - SEPIE



Software Engineering Process
Improvement Experiment



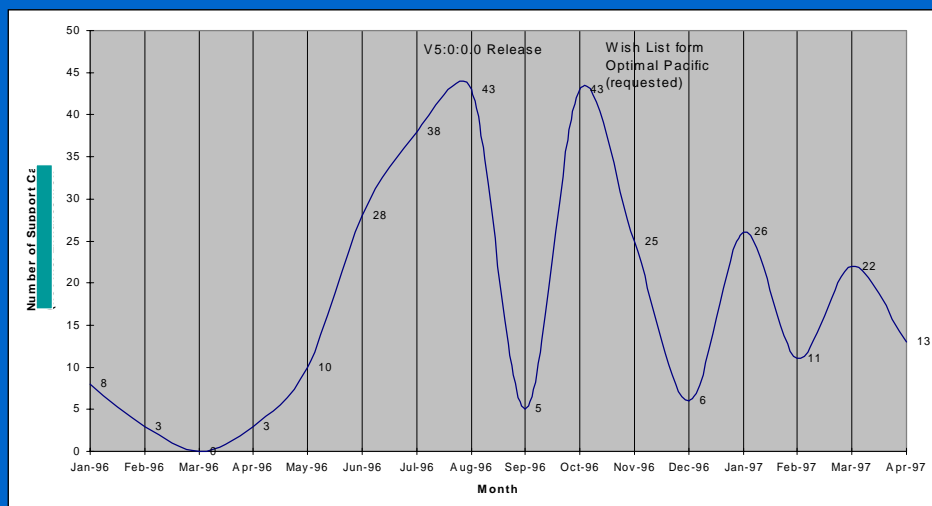
Who are Optimal Systems Ltd. ?

- Optimal Group in existence since 1985
 - Developers of Engineering & Survey applications
 - ISO 9001 & TickIT Certification
 - Third Party Software Developer's with Intergraph Corp, Bentley Systems, Autodesk Inc.
 - A team of professional engineers & surveyors
- 

What was the problem ?

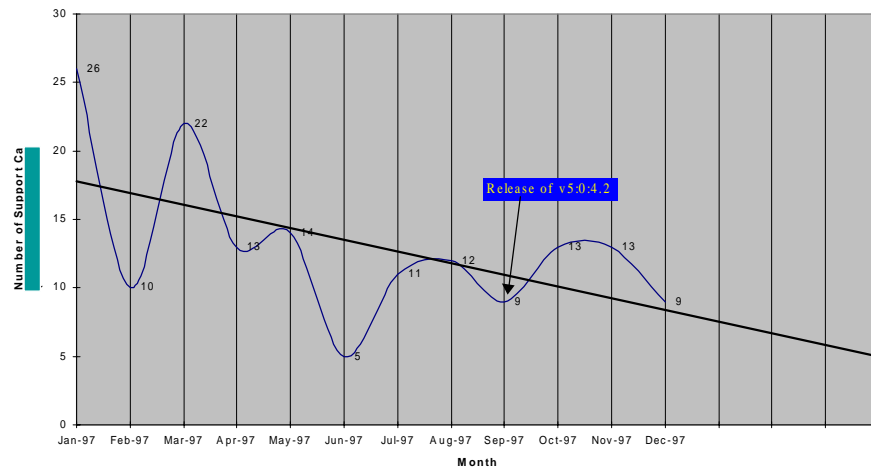
- Configuration Management - paper based system not working
- Geographic spread of agents
- Requirements management (compounded by email)
- Test Planning
- Information flow was base problem

Information Flow Problem



After QMS Improvements

Fig 1 : Trend Analysis



Selection of software tool

- PVCS Version Manager & Tracker
- Industry standard
- Review of other tools done prior to experiment



Implementation of VM v6.0

- Problems with VM v6
 - staff reaction
 - poor user interface
 - training absolute necessity
- Reaction to VM v6.5
 - very positive



Overall reaction to VM

- Experience with v6.0 regrettable
- Experience with v6.5 positive
- Functionality in both v6.0 & v6.5 seen as positive

Implementation of Tracker

- This is the main experimental area to improve information flow
- No problems initially
 - very positive reaction
- Problems experienced when upgrade received
- Overall an easy implementation

Project Results

- Change from VM v6.0 to v6.5 caused delay
- 20% saving on change request management
- Information flow has improved
- 30% time saving in creation of Test Plans
- 70% saving in creation of Release Notes
- 10 Man Days per programmer saved annually on Code Documentation
- 45% saving on disk space due to compression technology used by PVCS VM tool

Lessons Learned

- Do not under estimate number of licences required
- Complexity of Technology under estimated
- Hidden costs discovered (SQL Server)
- ESSI support required to give momentum to objectively assess contribution of technology

Lessons Learned

- Learning curve for tools are steep but must be overcome
- Poor user interfaces will not be accepted by staff
- Not all customers will be happy to be involved in experiments

•
•

Way Forward

- Continued use of PVCS Technology
- Positive impact on information flow
- PVCS will have a large impact on Quality System
- Metric collection to be revised for better management control

• • • • • • • •

WebSite Testing

Edward Miller
Software Research, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

© 1999 by Software Research, Inc.

Email comments to miller@soft.com
See also the companion paper [The WebSite Quality Challenge](#).

ABSTRACT

The instant worldwide audience of a WebSite's make its quality and reliability crucial factors in its success. Correspondingly, the nature of the WWW and WebSites pose unique software testing challenges. Webmasters, WWW applications developers, and WebSite quality assurance managers need tools and methods that meet their specific needs. Mechanized testing via special purpose WWW testing software offers the potential to meet these challenges. Our technical approach, based on existing WWW browsers, offers a clear solution to most of the technical needs for assuring WebSite quality.

BACKGROUND

WebSites impose some entirely new challenges in the world of software quality!

Within minutes of going live, a WWW application can have many thousands more users than a conventional, non-WWW application. The immediacy of the WWW creates immediate expectations of quality and rapid application delivery, but the technical complexities of a WebSite and variances in the browser make testing and quality control that much more difficult, and in some ways, more subtle, than "conventional" client/server or application testing. Automated testing of WebSites is an opportunity and a challenge.

DEFINING WEBSITE QUALITY & RELIABILITY

Like any complex piece of software there is no single, all inclusive quality measure that will fully characterizes a WebSite.

Dimensions of Quality. There are many dimensions of quality; each measure will pertain to a particular WebSite in varying degrees. Here are some common measures:

- *Timeliness:* WebSites change often and rapidly. How much has a WebSite changed since the last upgrade? How do you highlight the parts that have changed?
- *Structural Quality:* How well do all of the parts of the WebSite hold together? Are all links inside and outside the WebSite working? Do all of the images work? Are there parts of the WebSite that are not connected?
- *Content:* Does the content of critical pages match what is supposed to be there? Do key phrases exist continually in highly-changeable pages? Do critical pages maintain quality content from version to version? What about dynamically generated HTML (DHTML) pages?
- *Accuracy and Consistency:* Are today's copies of the pages downloaded the same as yesterday's? Close enough? Is the data presented to the user accurate enough? How do you know?
- *Response Time and Latency:* Does the WebSite server respond to a browser request within certain performance parameters? In an E-commerce context, how is the end-to-end response time after a **SUBMIT**? Are there parts of a site that are so slow the user discontinues working?
- *Performance:* Is the Browser->Web->WebSite->Web->Browser connection quick enough? How does the performance vary by time of day, by load and usage? Is performance adequate for E-commerce applications? Taking 10 minutes -- or maybe even only 1 minute -- to respond to an E-commerce purchase may be unacceptable!

Impact of Quality. Quality remains is in the mind of the WebSite user. A poor quality WebSite, one with many broken pages and faulty images, with Cgi-Bin error messages, etc., may cost a lot in poor customer relations, lost corporate image, and even in lost sales revenue. Very complex, disorganized WebSites can sometimes overload the

user.

The combination of WebSite complexity and low quality is potentially lethal to Company goals. Unhappy users will quickly depart for a different site; and, they probably won't leave with a good impression.

WEBSITE ARCHITECTURAL FACTORS

A WebSite can be quite complex, and that complexity -- which is what provides the power, of course -- can be a real impediment in assuring WebSite Quality. Add in the possibilities of multiple WebSite page authors, very-rapid updates and changes, and the problem compounds.

Here are the major pieces of WebSites as seen from a Quality perspective.

Browser. The browser is the viewer of a WebSite and there are *so* many different browsers and browser options that a well-done WebSite is probably designed to look good on as many browsers as possible. This imposes a kind of *de facto* standard: the WebSite must use only those constructs that work with the *majority* of browsers. But this still leaves room for a lot of creativity, and a range of technical difficulties. And, multiple browsers' renderings and responses to a WebSite have to be checked.

Display Technologies. What you see in your browser is actually composed from many sources:

- *HTML.* There are various versions of HTML supported, and the WebSite ought to be built in a version of HTML that is compatible. This should be checkable.
- *Java, JavaScript, ActiveX.* Obviously JavaScript and Java applets will be part of any serious WebSite, so the quality process must be able to support these. On the Windows side, ActiveX controls have to be handled well.
- *Cgi-Bin Scripts.* This is link from a user action of some kind (typically, from a FORM passage or otherwise directly from the HTML, and possibly also from within a Java applet). All of the different types of Cgi-Bin Scripts (perl, awk, shell-scripts, etc.) need to be handled, and tests need to check "end to end" operation. This kind of a "loop" check is crucial for E-commerce situations.
- *Database Access.* In E-commerce applications you are either building data up or retrieving data from a database. How does that interaction perform in real world use? If you give in "correct" or "specified" input does the result produce what you expect?

Some access to information from the database may be appropriate, depending on the application, but this is typically found by other means.

Navigation. Users move to and from pages, click on links, click on images (thumbnails), etc. Navigation in a WebSite often is complex and has to be quick and error free.

Object Mode. The display you see changes dynamically; the only constants are the "objects" that make up the display. These aren't real objects in the OO sense; but they have to be treated that way. So, the quality test tools have to be able to handle URL links, forms, tables, anchors, buttons of all types in an "object like" manner so that validations are independent of representation.

Server Response. How fast the WebSite host responds influences whether a user (i.e. someone on the browser) moves on or gives up. Obviously, InterNet loading affects this too, but this factor is often outside the Webmaster's control at least in terms of how the WebSite is written. Instead, it seems to be more an issue of server hardware capacity and throughput. Yet, if a WebSite becomes very popular -- this can happen overnight! -- loading and tuning are real issues that often are imposed -- perhaps not fairly -- on the WebMaster.

Interaction & Feedback. For passive, content-only sites the only real quality issue is availability. For a WebSite that interacts with the user, the big factor is how fast and how reliable that interaction is.

Concurrent Users. Do multiple users interact on a WebSite? Can they get in each others' way? While WebSites often resemble client/server structures, with multiple users at multiple locations a WebSite can be much different, and much more complex, than complex applications.

WEBSITE TEST AUTOMATION REQUIREMENTS

Assuring WebSite quality requires conducting sets of tests, automatically and repeatably, that demonstrate required properties and behaviors. Here are some required elements of tools that aim to do this.

Test Sessions. Typical elements of tests involve these characteristics:

- *Browser Independent.* Tests should be realistic, but *not* be dependent on a particular browser, whose biases and characteristics might mask a WebSite's problems.
- *No Buffering, Caching.* Local caching and buffering -- often a way to improve apparent performance -- should be disabled so that timed experiments are a true measure of the Browser-Web-WebSite-Web-Browser response time.
- *Fonts and Preferences.* Most browsers support a wide range of fonts and presentation preferences, and these should not affect how quality on a WebSite is assessed or assured.
- *Object Mode.* Edit fields, push buttons, radio buttons, check boxes, etc. All should be treatable in object mode, i.e. independent of the fonts and preferences.

Object mode operation is essential to protect an investment in test suites and to assure that test suites continue operating when WebSite pages experience change. In other words, when buttons and form entries change location on the screen -- as they often do -- the tests should still work.

However, when a button or other object is deleted, that error should be sensed! Adding objects to a page clearly implies re-making the test.

- *Tables and Forms.* Even when the layout of a table or form varies in the browser's view, tests of it should continue independent of these factors.
- *Frames.* Windows with multiple frames ought to be processed simply, i.e. as if they were multiple single-page frames.

Test Context. Tests need to operate from the browser level for two reasons: (1) this is where users see a WebSite, so tests based in browser operation are the most realistic; and (2) tests based in browsers can be run locally or across the Web equally well. Local execution is fine for quality control, but not for performance measurement work, where response time *including* Web-variable delays reflective of real-world usage is essential.

WEBSITE DYNAMIC VALIDATION

Confirming validity of what is tested is the key to assuring WebSite quality -- the most difficult challenge of all. Here are four key areas where test automation will have a significant impact.

Operational Testing. Individual test steps may involve a variety of checks on individual pages in the WebSite:

- *Page Consistency.* Is the entire page identical with a prior version? Are key parts of the text the same or different?
- *Table, Form Consistency.* Are all of the parts of a table or form present? Correctly laid out? Can you confirm that selected texts are in the "right place".
- *Page Relationships.* Are all of the links on a page the same as they were before? Are there new or missing links? Are there any broken links?
- *Performance Consistency, Response Times.* Is the response time for a user action the same as it was (within a range)?

Test Suites. Typically you may have dozens or hundreds (or thousands?) of tests, and you may wish to run tests in a variety of modes:

- *Unattended Testing.* Individual and/or groups of tests should be executable singly or in parallel from one or many workstations.
- *Background Testing.* Tests should be executable from multiple browsers running "in the background" on an appropriately equipped workstation.
- *Distributed Testing.* Independent parts of a test suite should be executable from separate workstations without conflict.
- *Performance Testing.* Timing in performance tests should be resolved to the millisecond; this gives a strong basis for averaging data.
- *Random Testing.* There should be a capability for randomizing certain parts of tests.
- *Error Recovery.* While browser failure due to user inputs is rare, test suites should have the capability

of resynchronizing after an error.

Content Validation. Apart from how a WebSite responds dynamically, the content should be checkable either exactly or approximately. Here are some ways that content validation could be accomplished:

- *Structural.* All of the links and anchors should match with prior "baseline" data. Images should be characterizable by byte-count and/or file type or other file properties.
- *Checkpoints, Exact Reproduction.* One or more text elements -- or even all text elements -- in a page should be markable as "required to match".
- *Gross Statistics.* Page statistics (e.g. line, word, byte-count, checksum, etc.).
- *Selected Images/Fragments.* The tester should have the option to rubber band sections of an image and require that the selection image match later during a subsequent rendition of it. This ought to be possible for several images or image fragments.

Load Simulation. Load analysis needs to proceed by having a special purpose browser act like a human user. This assures that the performance checking experiment indicates true performance -- not performance on simulated but unrealistic conditions. There are many "http torture machines" that generate large numbers of http requests, but that is not necessarily the way real-world users generate requests.

Sessions should be recorded live or edited from live recordings to assure faithful timing. There should be adjustable speed up and slow down ratios and intervals.

Load generation should proceed from:

- *Single Browser Sessions.* One session played on a browser with one or multiple responses. Timing data should be put in a file for separate analysis.
- *Multiple Independent Browser Sessions.* Multiple sessions played on multiple browsers with one or multiple responses. Timing data should be put in a file for separate analysis. Multivariate statistical methods may be needed for a complex but general performance model.

TESTING SYSTEM CHARACTERISTICS

Considering all of these disparate requirements, it seems evident that a single product that supports all of these goals will not be possible. However, there is one common theme and that is that the majority of the work seems to be based on "...what does it [the WebSite] look like from the point of view of the user?" That is, from the point of view of someone using a browser to look at the WebSite.

This observation led our group to conclude that it would be worthwhile trying to build certain test features into a "test enabled web browser", which we called CAPBAK/Web in the expectation that this approach would let us do the majority of the WebSite quality control functions using that engine as a base.

Browser Based Solution. With this as a starting point we determined that the browser based solution had to meet these additional requirements:

- *Commonly Available Technology Base.* The browser had to be based on a well known base (there appear to be only two or three choices).
- *Some Browser Features Must Be Deletable.* At the same time, certain requirements imposed limitations on what was to be built. For example, if we were going to have accurate timing data we had to be able to *disable* caching because otherwise we are measuring response times within the client machine rather than "across the web."
- *Extensibility Assured.* To permit meaningful experiments, the product had to be extensible enough to permit timings, static analysis, and other information to be extracted.

Taking these requirements into account, and after investigation of W3C's Amaya Browser and the open-architecture Mozilla/Netscape Browser we chose the IE Browser as our initial base for our implementation of CAPBAK/Web.

User Interface. How the user interacts with the product is very important, in part because in some cases the user will be someone very familiar with WebSite browsing and not necessarily a testing expert. The design we implemented takes this reality into account.

- *Pull Down Menus.* In keeping with the way browsers are built, we put all the main controls for CAPBAK/Web on a set of Pull Down menus, as shown in the accompanying screen shot.

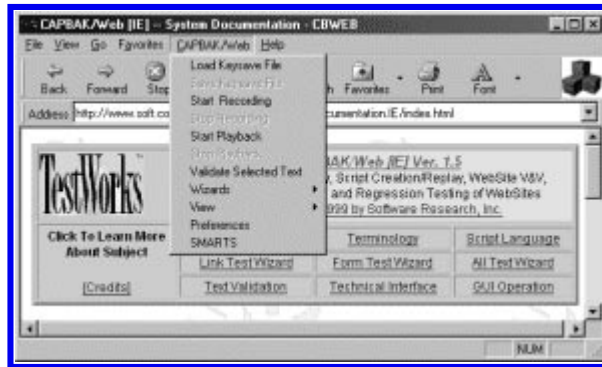


Figure 1. CAPBAK/Web Menu Functions.

- "C" Scripting. We use interpreted "C" language as the control language because the syntax is well known, the language is fully expressive of most of the needed logic, and because it interfaces well with other products.
- *Files Interface*. We implemented a set of dialogs to capture critical information and made each of them recordable in a text file. The dialogs are associated with files that are kept in parallel with each browser invocation:
 - *Keysave File*. This is the file that is being created -- the file is shown line by line during script recording as the user moves around the candidate WebSite.
 - *Timing File*. Results of timings are shown and saved in this file.
 - *Messages File*. Any error messages encountered are delivered to this file. For example, if a file can't be downloaded within the user-specified maximum time an error message is issued and the playback continues. (This helps preserve the utility of tests that are partially unsuccessful.)
 - *Event File*. This file contains a complete log of recording and playback activities that is useful primarily to debug a test recording session or to better understand what actually went on during playback.

Operational Features. Based on prior experience, the user interface for CAPBAK/Web had to provide for several kinds of capabilities already known to be critical for a testing system. Many of these are critically important for automated testing because they assure an optimal combination of test script reliability and robustness.

- Capture/Replay. We had to be able both to capture a user's actual behavior online, and be able to create scripts by hand.
- Object Mode. The recording and playback had to support pure-Object Mode operation. This was achieved by using internal information structures in a way that lets the scripts (either recorded or constructed) to refer to objects that are meaningful in the browser context.

A side benefit of this was that playbacks were reliable independent of the rendering choices made by the user. A script plays back identically the same independent of browser window size, type-font choices, color mappings, etc.

- [Adjustable] True-Time Mode. We assured realistic behavior of the product by providing for recording of user-delays and for efficient handling of delays by incorporating a continuously variable "playback delay multiplier" that can be set by the user.
- Playback Synchronization. For tests to be robust -- that is, to reliably indicate that a feature of a WebSite is working correctly -- there must be a built-in mode that assures synchronization so that Web-dependent delays don't interfere with proper WebSite checking. CAPBAK/Web does this using a proprietary playback synchronization method that waits for download completion (except if a specified maximum wait time is exceeded).
- Timer Capability. To make accurate on-line performance checks we built in a 1 millisecond resolution timer that could be read and reset from the playback script.
- Validate Selected Text Capability. A key need for WebSite content checking, as described above, is the ability to capture an element of text from an image so that it can be compared with a baseline value. This feature was implemented by digging into the browser data structures in a novel way (see below for an illustration). The user highlights a selected passage of the web page and clicks on the "Validate Selected Text" menu item.

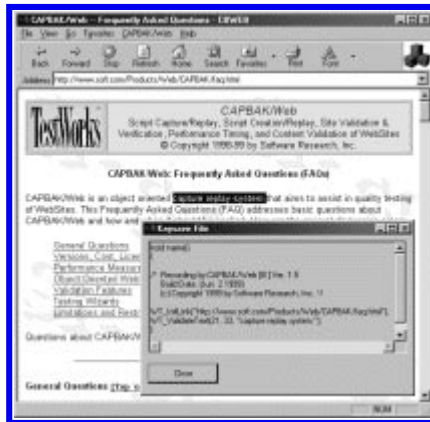


Figure 2. Illustration of CAPBAK/Web Validate Selected Text Feature.

What results is a recorded line that includes the ASCII text of what was selected, plus some other information that locates the text fragment in the page. During playback if the same text is **not** found at the same location an error message is generated.

- Multiple-playback. We confirmed that multiple playback was possible by running separate copies of the browser in parallel. This solved the problem of how to multiply a single test session into a number of test sessions to simulate multiple users each acting realistically.

Test Wizards. In most cases manual scripting is too laborious to use and making a recording to achieve a certain result is equally unacceptable. We built in several *test wizards* that mechanize some of the most common script-writing chores.

- *Link Wizard.* This wizard creates a script based on the current Web page that visits every link in the page. Scripts created this way are the basis for "link checking" test suites that confirm the presence (but not necessarily the content) of URLs.

Here is a sample of the output of this wizard, applied to our standard sample test page `example1.html`:

```
void name()
{
/* Produced by CAPBAK/Web [IE] Ver. 1.5 Link Wizard */
/* (c) Copyright 1999 by Software Research, Inc. */

WT_InitLink("http://www.soft.com/Products/Web/CAPBAK/example1/");
WT_GotoLink("http://www.soft.com/Products/Web/CAPBAK/example1/#target");
WT_GotoLink("http://www.soft.com/Products/Web/CAPBAK/example1/#notdefined");
WT_GotoLink("http://www.soft.com/Products/Web/CAPBAK/example1/example1.out.html");
WT_GotoLink("http://www.soft.com/Products/Web/CAPBAK/example1/example1.notoutside.html");
WT_GotoLink("http://www.soft.com/Products/Web/CAPBAK/example1/#topofpage");
}
```

Figure 3. Sample of Output of Link Test Wizard.

- *FORM Wizard.* For E-Commerce testing which involves FORMS we included in the system a FORM Wizard that generates a script that:
 - Initializes the form.
 - Presses each pushbutton by name.
 - Presses each radio button by name.
 - Types a pre-set script fragment into each text field.
 - Presses SUBMIT.

Here is a sample of the output of this wizard, applied to our standard test page: [example1.html](http://www.soft.com/Products/Web/CAPBAK/example1.html):

```

void name()
{
/* Produced by CAPBAK/Web [IE] Ver. 1.5 Form Wizard */
/* (c) Copyright 1999 by Software Research, Inc. */

WT_InitLink("http://www.testworks.com/Products/Web/CAPBAK/example1/");
WT_SubmitForm(FORM:0:12, "RESET FORM");
WT_SelectOneRadio(FORM:0:0, "now", "TRUE");
WT_SelectOneRadio(FORM:0:1, "next", "TRUE");
WT_SelectOneRadio(FORM:0:2, "look", "TRUE");
WT_SelectOneRadio(FORM:0:3, "no", "TRUE");
WT_SelectCheckBox(FORM:0:4, "concerned", "TRUE");
WT_SelectCheckBox(FORM:0:5, "info", "TRUE");
WT_SelectCheckBox(FORM:0:6, "evaluate", "TRUE");
WT_SelectCheckBox(FORM:0:7, "send", "TRUE");
WT_FormTextInput(FORM:0:8, "TestWorks");
WT_FormTextInput(FORM:0:9, "TestWorks");
WT_FormTextInput(FORM:0:10, "TestWorks");
WT_FormTextInput(FORM:0:11, "TestWorks");
WT_SubmitForm(FORM:0:13, "SUBMIT FORM");
}

```

Figure 4. Sample of Output of FORM Test Wizard.

The idea is that this script can be processed automatically to produce the result of varying combinations of pushing buttons. As is clear, the wizard will have pushed all buttons, but only the last-applied one in a set of radio buttons will be left in the TRUE state.

- *Text Wizard.* For detailed content validation this wizard yields up a script that includes in confirmation of the entire text of the candidate page. This script is used to confirm that the content of a page has not changed (in effect, the entire text content of the subject is recorded in the script).

EXAMPLE USES

Early application of the CAPBAK/Web system have been very effective in producing experiments and collecting data that is very useful for WebSite checking. While we expect CAPBAK/Web to be the main engine for a range of WebSite quality control and testing activities, we've chosen two of the most typical -- and most important -- applications to illustrate how CAPBAK/Web can be used.

Performance Testing Illustration. To illustrate how CAPBAK/Web measures timing we have built a set of *Public Portal Performance Profile* TestSuites that have these features:

- *Top 20 Web Portals.* We selected 20 commonly available WebSites on which to measure response times. These are called the "P4" suites.
- *User Recording.* We recorded one user's excursion through these suites and saved that keysave file (playback script).
- *User Recording.* We played back the scripts on a 56 kbps modem so that we had a realistic comparison of how long it would take to make this very-full visit to our selected 20 portals.
- *P4 Timings.* We measured the elapsed time it took for this script to execute at various times during the day. The results from one typical day's executions showed a playback time range of from 457 secs. to 758 secs (i.e. from -19% of the average to +36% of the average playback time).
- *Second Layer Added.* We added to the base script a set of links to each page referenced on the same set of 20 WebSites. This yielded the P4+ suite that visit some 1573 separate pages, or around 78 per WebSite. The testsuite takes around 20,764 secs (~5 Hrs 45 mins) to execute, or an average of 1038 secs per WebSite. per WebSite).
- *Lessons Learned.* It is relatively easy to configure a sophisticated test script that visits many links in a realistic way, and provides realistic user-perceived timing data.

E-Commerce Illustration. This example shows a typical E-Commerce product ordering situation. The script automatically places an order and uses the Validate Selected Text sequence to confirm that the order was processed correctly. In a real-world example this is the equivalent of (i) selecting an item for the shopping basket, (ii) ordering it, and (iii) examining the confirmation page's order code to assure that the transaction was successful. (The final validation step of confirming that the ordered item was actually delivered to a specific address is not part of what CAPBAK/Web can do.)

- *Example Form.* We base this script on a sample page shown below. This page is intended to have a form that shows an ordering process. On the page the "Serial Number" is intended as a model of a credit card number.

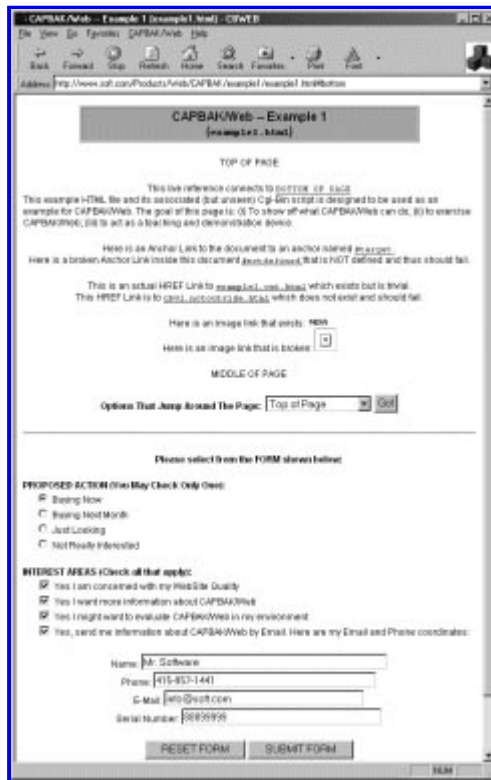


Figure 5. Sample Input Form For E-Commerce Example.

- *Type-In with Code Number.* Starting with the FORM Wizard generated script, we modify it to include only the parts we want, and include the code number 8889999.
- *Response File.* Once the playback presses the SUBMIT button the WebServer response page shows up, as shown below.



Figure 6. Response Page for E-Commerce Example.

- *Error Message Generated.* If the Cgi-Bin scripts make a mistake this will be caught during playback because the expected text 8889999 will no be present.
- *Completed TestScript.* Here is the complete testscript for CAPBAK/Web that illustrates this sequence

of activities.

```
void name()
{
  /* Recording by CAPBAK/Web [IE] Ver. 1.5
     (c) Copyright 1999 by Software Research, Inc. */

  WT_InitLink("http://www.soft.com/Products/Web/CAPBAK/example1/example1broken.html");
  WT_SelectOneRadio(FORM:1:0, "buying-now", "TRUE");
  WT_SelectOneRadio(FORM:1:1, "next-month", "FALSE");
  WT_SelectOneRadio(FORM:1:2, "just-looking", "FALSE");
  WT_SelectOneRadio(FORM:1:3, "no-interest", "FALSE");
  WT_SelectOneRadio(FORM:1:4, "Yes", "TRUE");
  WT_SelectOneRadio(FORM:1:5, "Yes", "TRUE");
  WT_SelectOneRadio(FORM:1:6, "Yes", "TRUE");
  WT_SelectOneRadio(FORM:1:7, "Yes", "TRUE");
  WT_FormTextInput(FORM:1:8, "Mr. Software");
  WT_FormTextInput(FORM:1:9, "415-550-3020");
  WT_FormTextInput(FORM:1:10, "info@soft.com");
  WT_FormTextInput(FORM:1:11, "8889999");
  WT_SubmitForm(FORM:1:13, "SUBMIT FORM");
  WT_Wait(3425);
  WT_ValidateText(12, 143, "8889999");
}
```

Figure 7. Script for E-Commerce Test Loop.

- *Lessons Learned.* This examples illustrates how it is possible to automatically validate a website using CAPBAK/Web by detecting when an artificial order is mis-processed.

FUTURE EXPANSION AND EXTENSIONS

We are presently using CAPBAK/Web in support of various customers' WebSite quality control activities. Yet, as rich as we believe our implementation of a test enabled web browser is with CAPBAK/Web, there are many areas where there is need for expansion.

Obviously we need to expand the capability to the Mozilla class of browsers, and possibly others as well. And, certain of the user-control functions have to be refined to get the best use out of the product set.

SUMMARY

All of these needs and requirements impose constraints on the test automation tools used to confirm the quality and reliability of a WebSite. At the same time they present a real opportunity to amplify human tester/analyst capabilities. Better, more reliable WebSites should be the result.

REFERENCES

This paper is based on many sources; it relies heavily on a prior White Paper [The WebSite Quality Challenge](#).

You can learn more about the test system being described in a [Tour of CAPBAK/Web](#).

There is detailed information about the [P4 Examples](#) and the [P4+ Examples](#).

You can study the E-commerce example described above by going to these URLs:

[example1.html Input Page](#) :
[example1.html Response Page](#):

Readers may also be interested in seeing one way that the CAPBAK/Web product is applied by considering subscribing to one of the family of [eValid Test Services](#).

The Impact on Staff of the Implementation of PVCS Technology

Author: Dermot Hore
Project Manager
Company: Optimal Systems Ltd
Molesworth House
1 South Frederick Street
Dublin 2
Ireland.
Email: dermot.hore@optimal.ie
Phone: +353 1 6779555
Fax: +353 1 6779642

Related European Project: ESSI Project number 27469
Acronym : SEPIE (Software Engineering Process Improvement Experiment)

Abstract

Optimal Systems provides a comprehensive solution for interactive modelling, design, and analysis of transmission and distribution lines. Through regular surveillance audits, information flow has been identified as the underlying cause of a number of problems being experiencing. These include project overruns, continual requirements change, and errors in transition from change requests/bugs received via email too, requirements specification and onwards to system test.

The experiment will assess the performance of PVCS Version Control and Tracker modules to improve the information flow between the agents and development staff and also among the development staff.

The experiment, funded by the European Commission, will test the expectation that for small to medium size software development companies, improved information flow, through the use of software tools, will improve the quality of the end product, gain better control over project effort and improve effort estimation and scheduling.

Introduction

This paper sets out the experiences of the staff of Optimal Systems Ltd in using PVCS (Tracker and Version Manager modules) to support improvements in communication flow within the organisation. The process was supported by ESSI as the SEPIE project, Number 27469. Optimal Systems Ltd is a Dublin based company that is engaged in the design, development and delivery of application software for the Electrical Supply Industry worldwide. Positive staff reaction is critical to the successful implementation of new technologies.

The process improvement experiment on which this paper is based is an 11-month project that commenced in July 1998 and was completed in June 1999. This paper focuses on the impact on staff on the use of the new technology and their assessment of how it contributes to improving the development process.

Background

Optimal Systems is typical of many small software development organisations insofar as there is a strong dependency on individuals and there is a tendency to informality in the modification of requirements during the development process. Optimal also has a very dispersed agent base who provide input to the requirements gathering and approval process. At the time the experiment commenced a number of issues had been identified that were of major concern:

- Code error accounted for 18% of defects and indicated the need for improvements in user and acceptance testing. These difficulties were arising in part from the failure to generate appropriate test plans from the requirement specification.
- There were poor control over requirement changes with some requests sent directly to the developer with a know negative impact on version control
- Change requests were sent by email but there was no system for logging and linking similar requests into a formal requirements specification. This has a negative impact on the effort required to create accurate requirements specification.
- The level of effort required from the completion of testing to product release was a major contributor to project overruns.

The expectation was that the implementation of PVCS modules would have a positive impact on each of these issues resulting in:

- Improved Information flow
- More comprehensive and efficient testing
- More consistent and thorough requirements gathering
- Reduced project overruns

Approach

The experiment was broken down into three distinct phases. During the first phase of the project the PVCS modules were procured and training received. In addition historical data within the organisations was assessed in order to facilitate the objective assessment of the contribution of PVCS to the software development process.

The second phase is the main data gathering phase of the project against which the performance of PCVS modules will be evaluated from both technical and staff perspectives.

The third phase is concerned with the comparative data analysis.

A key component of the assessment of PVCS is the impact on staff. Staff acceptance was regarded as critical to successful implementation. In order to assess staff reaction throughout the experiment regular workshops were held with staff and also an attitude questionnaire was developed and distributed at key points. The remainder of this paper concentrates on this data.

Overview of PVCS

Version Manager Module

This is designed for organising, managing and protecting software assets. Version Manager enables teams of any size and location to co-ordinate concurrent development with secure access and complete audit trail. Version Manager claims to improve quality and accelerate team development by eliminating problems caused by lost changes, overwrites and content errors and by increasing reuse and automating common tasks. Within Version Manager you can

- a) Help document and control change through multiple revisions
- b) Trace changes and allow parallel development
- c) Secure archives and all project code and documentation.

Tracker Module

This module is design to manage technical and business issues. It captures, manages and communicates feature requests, defect reports and changes. Within Tracker you can

- a) Track enhancement requests, change requests and defect reports
- b) Establish priorities, assign ownership, manage hand-offs and track issues through their lifecycle
- c) Automatic notification is possible when ownership of an issue is assigned.

Staff Reaction to PVCS Following Training

The initial staff reaction was obtained by questionnaire following training. This questionnaire was broken into the following sections:

- Overall reaction to the Module Capabilities
- Screen Layout
- Terminology
- Learning

From the outset it was clear that the reaction to the Version Manager was radically different to that of Tracker. This reflects the different level of development of the two modules and their interface style.

Overall reaction to the Module Capabilities

The Tracker module was rated very positively at the end of the training session. This is the more modern component of the PVCS suite. The reaction to Version control was very poor in particular the staff found it

difficult to know which options they were to choose from. Neither did they find the system satisfying to use.

Despite their negative reaction to the interface however they did still recognise that the application functionality would meet the needs of Optimal.

Module Capabilities

Two topics were addressed in relation to module capabilities. These were speed and the rate of display of information. The key area of concern in relation to speed was the time required to check files in and out. The staff voiced concern over the time it would take to carry out this activity in a real world scenario rather than the sample cases used in training. Following technical support from Intersolve Inc. (suppliers of PVCS) it was discovered that the virus protection software in combination with the PVCS modules was the source of the speed problem. Removing some of the virus protection enabled the PVCS modules to operate at an acceptable speed.

Screen Layout

The screen layout is a key element in supporting the user in their interaction with an application. This is an area where the PVCS Version Manager module received very poor ratings. From the participants comments it would seem that while the logic/process flow seemed ok the level of feedback to the user was poor and so they had difficulty in determining whether or not they had engaged in the correct action.

The poor screens may cause potential problems in future. In the short term, all users of Version Manager module should determine whether they need to establish a “how to” list for the activities that they will be using regularly to support them in their tasks.

It is likely however that the poor nature of the interface will discourage individuals from exploring the full potential of the system.

Terminology

The poor rating of error messages is a cause for concern as it is reasonable to expect that in a system as large and as complex as PVCS that users will make mistakes. The difficulty seems to be that the users ended up in places where they didn't want to be. However they did acknowledge that the navigation process itself was quite clear. One user comment deserves particular mention:

“You always know where you were.... You may not have wanted to be there”

Taken together with the poor ratings on screen layout there is a high risk that the Optimal staff will find these modules difficult to use on an ongoing basis and this may have a negative impact on their adoption of the technology. Every effort should be made in the short term to concentrate on a small number of core interactions that the staff learn well.

Learning

Despite the negative responses to earlier sections the staff did feel that the modules were learnable, although there were some difficulties to be overcome. There was an acknowledgement that training in the application was essential. However for an organisation of the size of Optimal this was a realistic goal. The main reservation of the staff was on the poor quality of the user interfaces rather than the functionality of the system that was perceived positively.

Staff Reaction Following the Input of Historical Data

Version Manager v6.0

The first module to be implemented was the Version Manager Module. Having completed the task of implementing the data from an historical project the staff reaction was very bi-polar. At one extreme the amount of functionality was recognised as being very extensive and extremely useful to the user. At the opposite end the interface seems to be linked on to a legacy system and very difficult to interact with.

Version Manager v6.5

It is acknowledged that the new updated interface (v6.5) is now available. The new update was delivered and reaction is positive. However most of the users within Optimal would want to run the product in the background and the new version does not include any changes to the SCC interface used within specific development environments such as Microsoft Visual Studio.

Tracker

The Tracker module was implemented to handle email. As a change control system it is working well. The new interface contained in this module is considered more positively than the Version Manager interface. The module is having a positive impact on the effort expended recording and tracking customer issues both during a project life cycle and general maintenance issues.

Currently we are investigating the possibility of standardising the format of the submission of customer issues in order to reduce the effort of data input to allow more time to be allocated to issue resolution.

Project Results

- Change from VM v6.0 to v6.5 caused delay
- 20% saving on change request management
- Information flow has improved
- 30% time saving in creation of Test Plans
- 70% saving in creation of Release Notes
- 10 Man Days per programmer saved annually on Code Documentation
- 45% saving on disk space due to compression technology used by PVCS VM tool

Lessons Learned

- Do not under estimate number of licences required
- Complexity of Technology under estimated
- Hidden costs discovered (SQL Server)
- ESSi support required to give momentum to objectively assess contribution of technology
- Learning curve for tools are steep but must be overcome
- Poor user interfaces will not be accepted by staff
- Not all customers will be happy to be involved in experiments

Way Forward

PVCS is a very complex product with tremendous capability. There was a strong need for the interface to be overhauled to better support user interaction, v6.5 of the Version Manager module has addressed this issue. The negative reaction to the Version Manager v6.0 experienced by the Optimal staff following training and initial implementation has resulted in the staff avoiding using the product.

The Tracker module on the other hand is being used successfully within the organisation and is making a positive contribution to the quality issues recognised by the organisations at the start of this technology experiment. It is hoped to automate the data input to further improve the use of this module.

Test tool selection

by
Mark Paap

m.j.paap@iquip.nl

Agenda

- Introduction
- The test tool selection process
- Aspects that need special attention
- Preparing for implementation
- Real-life examples

Introduction

Why do organizations buy test tools?

- + to optimize the testing process
- + to decrease lead time of test execution
- + to decrease the need for human resources
- + to improve quality of testware and test execution
- + to standardize the products of the testing process

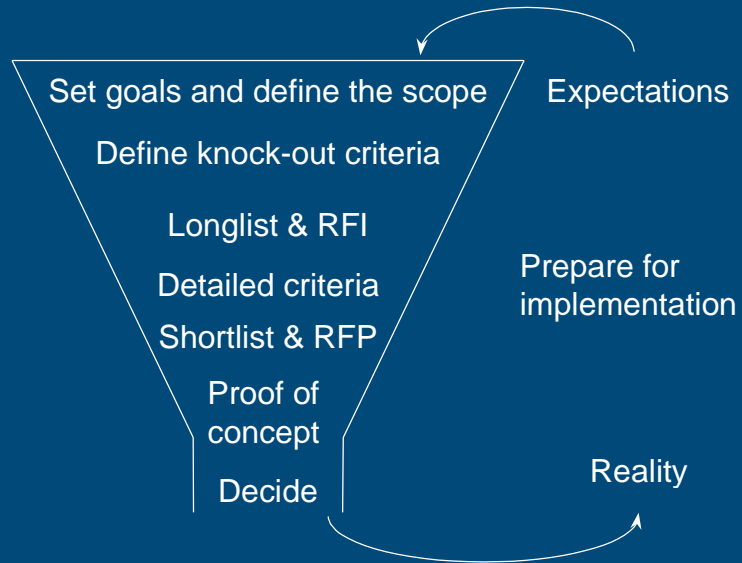
- to avoid (manual) testing
- to solve organizational problems

Introduction

What is the problem with test tool selection?

- current test process not structured
- unrealistic expectations
- no clear goals and scope
- done for the wrong reasons
- no knowledge of test tools
- no experience in selecting tools

The test tool selection process (1/5)



The test tool selection process (2/5)

- Set goals and define the scope
- Define knock-out criteria
 - hardware and operating system platform
 - development tools used
 - runtime environment of system under test
- Longlist & RFI
 - knock-out criteria
 - questions (explicit, business case, product information, open invitation)



The test tool selection process (3/5)

- Detailed criteria
 - interviews
 - documentation
 - build an evaluation team
- Shortlist & RFP
 - send detailed criteria to vendors
 - demos and reference sites
 - evaluate proposals
 - make a choice



Real-life example

Demo sessions on site

- AS/400, Terminal emulator, financial applications
- TPI®, evaluation team (4 members)
- well prepared demo sessions on site (4)
- 2 tools remained
- choice based on :
 - support of Terminal emulator
 - response on questions
- proof of concept resulted in guide lines
- implementation by evaluation team members

The evaluation team

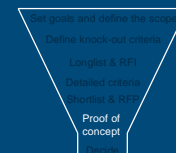
- Consists of
 - future users of the tool
 - support representative
 - (future) champions and change agents
 - consultant

- Activities
 - prepare demos and visit reference sites
 - evaluate proposals
 - make a choice
 - perform proof of concept
 - prepare for implementation

The test tool selection process (4/5)

- Proof of concept
 - user acceptance test of the tool
 - try and learn
 - collect metrics to quantify the benefits

- For Capture & Playback tools investigate
 - synchronization
 - object recognition
 - data driven
 - checking
 - error recovery





Real-life example

Customer rule the waves

- tool vendor promised a 35 % cost reduction for Y2K testing (est. 80 man years)
- vendor was invited for proof of concept
- customer selected tests to automate on strategic platforms (4)
- during proof of concept, metrics were collected
- savings were less than 15 %
- customer decided to optimize the usage of already available tools with already available resources



The test tool selection process (5/5)

- Decide, based on
 - criteria
 - proof of concept findings
 - expected return on investment
 - goals

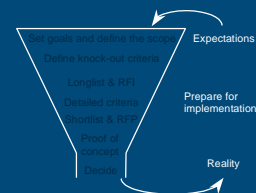


Aspects that need special attention

- integration with the existing test process
- integration with other tools
- split between technical and non-technical users
- maintenance of test products
- support for different environment(s)
- object recognition (GUI, Terminal emulator, browser)
- synchronization

Preparing for implementation

- Take care of
 - expectations & reality
 - support organization
 - guidelines
 - pilot project
- Communicate in order to
 - justify the choice
 - bridge the gap between expectations and reality
 - (re)set goals and scope of tool usage

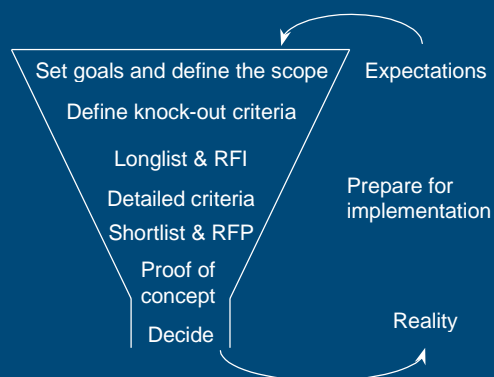


Real-life example

Architecture of an integrated test suite

- customer wishes to obtain an integrated testsuite that can support the overall testing process
- current test tool usage was investigated
- for each missing type of test tool a selection process
- the proof of concept was done on the integrated test suite
- integration based on an architecture
- architecture became the key to assembling and implementation
- preparing for implementation was a major effort

Questions ?



Managing the E-Business Operations

Dr Fawzy Soliman

School of Management,

University of Technology, Sydney

P.O. Box 123, Broadway, Sydney, NSW, 2007, AUSTRALIA.

Fawzy.Soliman@uts.edu.au

ABSTRACT

In the past few years, the number of Internet users has growing very steeply with most of this growth coming from newly registered commercial enterprises. The Internet is an effective method for shortening the development cycle of new products, communicating with suppliers and experts from around the world, receiving customer feedback, and accessing supercomputers for research and development. Advances in the Internet technology have created an information infrastructure for moving information between and within organizations.

Conducting business on the Internet is a complex process, which can be divided into three evolutionary stages. The first stage begins with a Corporate Web-site presence. The second stage known as the Electronic Commerce (E-Commerce) stage involves more business activities than the first stage. In the third stage companies can use the internet for a full scale business transactions. This stage is known as the Electronic Business (E-Business) stage.

Increased competition and globalization in the business arena have had a major impact on business organizations and have forced many business operations all around the world to move away from the traditional methods of communication to the Internet driven Electronic Business (E- Business) Operations.

This paper presents a model for the management of E- Business Operations and highlights the key driving forces for the adoption of E-Business. In addition, the paper analyses the economic values, and strategic benefits of using Internet-based E- Business operations. The model, depicts how companies can develop and implement Internet E-Business to support their operations.

Key Words: Internet, Electronic Commerce, Competitive Advantage, Economic Value.

1. Introduction

Business operations are increasingly facing unstable business environment which is influenced by fast advances in information technology (IT). These changes have occurred due to increased global competition, significant advances in international business, and changes in political and economical environments around the world.

The Internet Technology has been developing very rapidly and as such it is difficult to predict the level and extent of Internet usage in business operations. However, expert predictions show that Internet-Based Business Operations already started to change the way of conducting business.

Business Operations over the Internet are very much in their infancy. They are rapidly becoming the new method to conduct business and to interact with customers, suppliers and partners. Electronic Business Operations cover many aspects of buying/selling relationships and also many operations within manufacturing.

According to Soliman and Gide (1997), "over the next ten years, the growth of Internet-Based E-Commerce will outstrip the growth of traditional commerce. It is the commercialization of the Internet that is leading the way to this remarkable growth in Electronic Business Operations".

There has been a phenomenal growth in commercial presence on the Internet in recent times. In the last 2 years the commercial domain registrations of the entire Internet have grown to represent some 85% of all organizations. This effectively kills the myth that the Internet is an academic and research playground. Facts and figures from industry analysis show that (Gide and Soliman, 1997a):

1. Internet-Based E-Business is expected to reach \$150 billion by the year 2000 and more than \$1 trillion by the year 2010. Sales generated via the Web have grown from \$17.6 million in 1994 to nearly \$400 million in 1995 (a growth rate of over 2100%). The number of sites using the Internet for product transactions has increased from 14% in 1995 to 34% in 1996 and to a projected increase of 44% in the next 3 years.
2. The Internet has reduced the number of letters, voice calls and faxes around the globe. Thirty per cent of Internet users in one survey stated that Internet usage had resulted in new business opportunities and 43% said that it has increased productivity (Soliman, 1998).

According to ActivMedia (1997), projections indicate that global Web sales through 2002 could total \$1.5 trillion, or about 3% of combined Gross Domestic Product (GDP) for all countries worldwide. The study tracked eight Web business segments: manufacturing, computers and software, business and professional, consumer, travel, investment/finance, publishing, and real estate. In addition the market research firm Paul Kagan and Associates released 10-year revenue projections for the interactive media industry, showing that in the year 2007, the Internet-related income is expected to be \$46 billion, having risen from a projected \$11.1 billion for 1997. Electronic Commerce, revenue is expected to increase from \$0.9 billion in 1997 to \$11.7 billion over the next 10 years.

Books and computer hardware and software are the items most people purchase most via the Web, according to data from the most recent study of Internet demographics by Nielsen Media Research and Industry Trade Association CommerceNet. The study shows that 5.6 million people have purchased books online, while 4.4 million people have purchased hardware, and 4 million people have purchased software via the Internet.

According to Nielsen Media Research and CommerceNet, 78 million people used the Web during the first six months of 1998, and 20 million of those users made purchases via the Web. The following are the highlights of shopping and purchasing activities from a recent study (Gide and Soliman, 1998):

- ❑ 48 million Web shoppers - increase of 37% from September 1997.
- ❑ 20 million Web purchasers - increase of 100% from September 1997.
- ❑ 71% of Web purchasers are men, 29% are women - unchanged from September 1997.
- ❑ Women represent 36% of all online book buyers and 12% of all online computer hardware buyers.
- ❑ Among persons age 16-24, the top items purchased on the Web are books, CDs/cassettes/videos, and clothing.
- ❑ Among persons 50 years and older, the top items purchased on the Web are books, software and computer hardware.
- ❑ Consumers under the age of 35 represent 65% of all persons buying clothing on the Web, and 64% of all persons buying CDs/cassettes/videos.
- ❑ Consumers 35 years old and over represent 63% of all persons buying computer hardware on the Web, 59% of all persons buying software and 58% of all persons buying books.

Businesses are aggressively adopting inter-company trade over the Internet because they want to cut costs, reduce order-processing time, and improve information flow (Cronin, 1996b). For most firms, the rise in trade over the Internet also coincides with a marked decrease in telephone and facsimile use, allowing salespeople to concentrate on pro-actively managing customers' accounts rather than serving as information givers and order takers.

The Internet and its application tools have led to a global business information infrastructure, which now rivals the conventional telephone systems in size, coverage and popularity. As the commercial use of the Internet grows, it is becoming increasingly recognized that this is a very different business environment from its physical counterpart. According to Hoffman and Novak (1996), marketing approaches, which work well for broadcast or print media, may not perform as well on the Internet. Common ways of exploiting the Internet as a business tool include marketing and information distribution; electronic mail for inter-company communication; and provision of services and products. Rayport and Sviokla (1995) also suggested that

businesses could use the Internet to help them gain access to marketplaces (or “marketspaces”), which might otherwise be inaccessible.

It is widely accepted that companies are using the Internet because they believe they can gain competitive advantage. Poon and Swatman (1995) suggested that integration of Internet usage with business strategy is critical to its success. They also reported that the amount of and reasons for integration, or lack of it, are still unknown.

During the last two decades, many companies adopted the Electronic Data Interchange (EDI) technology to enter into the paperless economy. With the passage of time this has changed and now experts are debating whether businesses will abandon the well structured, and planned EDI processes in favor of E-Business. Many businesses choose EDI as a fast, inexpensive and safe method of sending purchase orders, invoices, shipping notices and other frequently used business documents.

2. What Electronic-Business Operation is?

Electronic Commerce, Electronic Trading and Electronic Business are often used interchangeably and many times there is a perception that these terms principally refer to the procurement cycle - the ordering and paying for goods or services either via electronic commerce technologies such as EDI or, more recently and growing in popularity, on-line Internet shopping.

Internet-Based E- Business is not an extension of EDI (Electronic Data Interchange) which has been primarily limited to computer-to-computer transactions, and has not been associated with major transformations of firms. However, the Internet-Based E-Business is giving a new way to conduct Business Operations, with different characteristics and is an evolution from EDI (Soliman and Gide, 1997).

There is no exact definition of E-Business Operations. Since, Internet commerce is still immature, so is the definition. However, one definition made by Kalakota (1996), as *into value- added outputs*”. Basically, this process involves taking information as raw material and producing value added information-based products or services out of the original raw information (Soliman, 1998).

E-Business Operations refers to an on-line production process owned by intermediaries. Producers of information interact with services and other processed information, such as orders, payments or instructions. In reality, Internet Business Operation is about businesses and consumers adopting a new process or methodology in dealing with each other. These processes are in essence supported by electronic interactions that replace close physical presence requirements or other traditional means(Gide and Soliman, 1997b). Basically, the Internet E-Business process involves taking information as raw material and producing value added information-based products or services out of the original raw information as shown in the following figure (Figure 2).

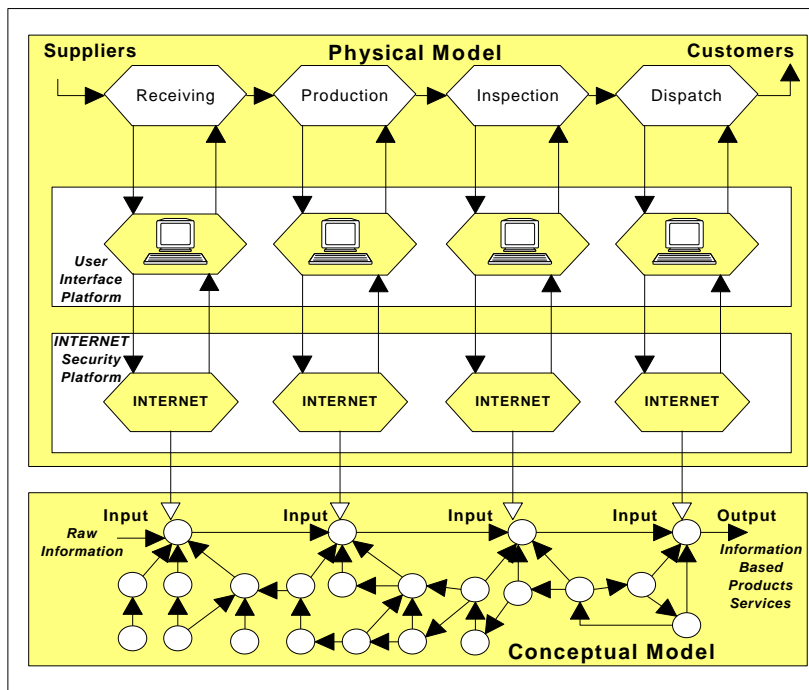


Figure 2: Illustration of the E-Business as a value Process.

The above figure shows two models for E-Business Operations (*Physical and Conceptual*) Models. In the Physical Model raw material enters the system and leaves as finished goods. In the Conceptual Model the Internet operates through two platforms (*User and Security Platforms*) here raw data enters the system and leaves as processed information.

Producers of information interact with services and other processed information, such as orders, payments or instructions. In reality, Internet E-Business operation is about businesses and consumers adopting a new process or methodology in dealing with each other. These processes are supported by electronic interactions that replace close physical presence requirements and traditional means.

Some authorities are already claiming that the main benefit of the Internet to-date is better customer service. However, there is a wide debate about where future investments will be made, that is: a) to support business-to-business processes, or b) to back office processes. The technologies and products which will enable businesses to do business with each other over the Internet is generally agreed to be attracting between 5 and 8 times the near-term future investment.

3. Key Values of Electronic Business Operations

There are various types of key measurements that must be tracked prior to embarking on a full implementation. Some of the important key elements to measure business value are:

- **Improving customer service:** Providing customers self-access to their accounts, transactions and orders, is a valuable service. The level of satisfaction for those customers interacting electronically will undoubtedly rise.
- **Reducing costs:** The most basic cost reductions could be related to publishing costs, which include the cost of production, printing and distribution. Furthermore, marketing and selling costs are also lower in an electronically enabled business environment.
- **Providing Business Intelligence:** In the Electronic Business Operations world, businesses need to know much more about their clients. Electronic commerce makes it possible to market to specific individuals based on their patterns of (purchasing and browsing) behavior. Hence they need to capture, and to analyze, as much information as possible about each individual purchase (or cancelled purchase) in order to build up customer's profiles. This is achieved in much the same way that neighborhood stores once did, through personal acquaintance with the consumer and continuous contact. The use of this analyzed data leads to what is being called "market response" systems or "adaptive marketing".

- **Process simplification:** Instead of using paper, using the World Wide Web (WWW) simplifies and speeds the approval process.
- **Generating new revenue:** The new Internet-Based Electronic Marketplace generates new revenue by selling new products and services specifically designed for the electronic marketplace. Existing product or services can also be sold on the Internet.
- **Taking faster decisions:** By receiving information about competition through an Intranet information retrieval database, it would be possible to develop a competitive strategy much faster than otherwise. The drivers for business are customer's needs and time. Time is a major source of competitive advantage and competitive pressures requiring production schedules to be shortened.

4. Types of Internet E-Business Operations

At present, there are three types of Internet Business Operations, these three types are:

- Business to Business
- Business to Consumer, and
- Business to Employee.

Business-to-Business Operation is complementary to EDI in that it is beginning to be used for non-production, non-replenishment applications. The widely used current terms used to describe the function of Electronic Business Operations are "Business to Business" and "Business to Consumer". The expression "business-to-business" is inexact and sometimes misleading. In Electronic Business Operations it is not always possible to tell who is accessing the automated point of sale/point of contact. It could be a retail consumer buying in wholesale quantities; it could be a business buying in retail quantities-and many other variants. Business-to-Business automated ordering processes are designed to empower business managers. The business server can only be accessed through the corporate Intranet, or an Extranet for "communities of interest".

Businesses-to-business operations involve companies and their suppliers while consumer markets include home shopping, banking, health care and broadband--or high-power--communications to the home.

In manufacturing, traditionally Design Engineering, Procurement and Production Departments communicate with each other using paper based methods. However the introduction of Internet-Based Electronic Business Operations and its superiority of over traditional EDI is adding new dimension to reducing the cost of manufacturing. So, in a typical manufacturing setting Design Engineering Department supply design drawings and specification to Procurement Department to procure material, commence production, and ultimately deliver goods to customers as per orders. In a general manufacturing setting, there are three types of flows: a) Material flow, b) Clerical flow and c) Information flow. Improvement in the movement of raw material, Work-In-Process and Finished Goods is likely to occur as a result of using the Internet-Based Electronic Business Operations. The main benefit to manufacturing from the Internet lies in the second and third types of flow.

The number of parts used in production could be in the order of thousands of items. These parts are usually purchased from suppliers on the basis of price, quality, and delivery on time and suppliers financial position and reputation in the industry.

Accordingly Material Procurement professionals must be equipped with timely and valuable information on parts and their suppliers. The Internet-Based Electronic Business Operations provide them with a fast and efficient way of obtaining comprehensive information of the market, feedback from the industry and the performance of suppliers.

The following figure (Figure 3) illustrates how clerical and production information can be efficiently and cost-effectively communicated throughout the supply chain using the Internet-Based Electronic Business Operations.

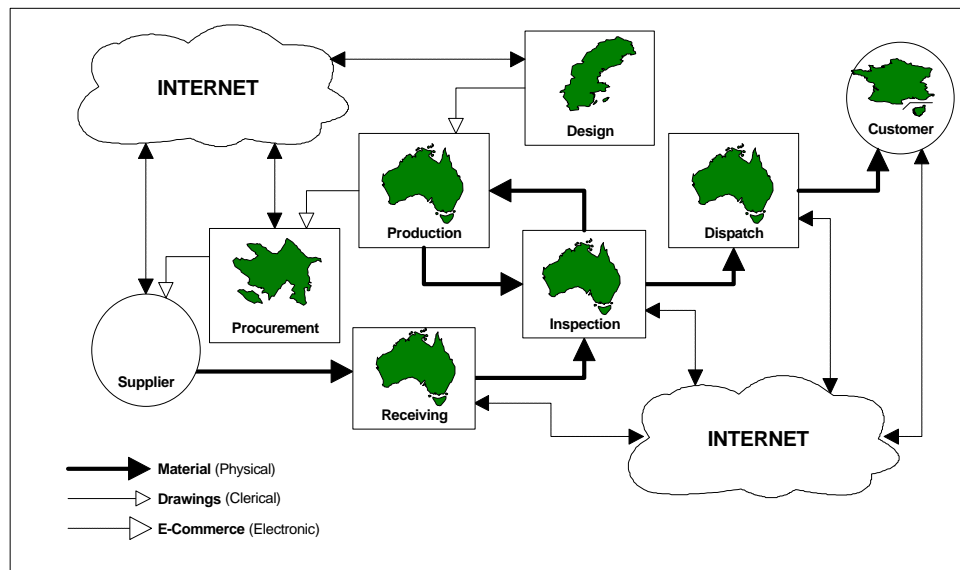


Figure 3: Supply Chain Communication in Manufacturing using Internet-Based Electronic Business Operations.

The Business-to-Consumer Operation complements normal retail shopping, mail order and direct marketing. It can accommodate delivery of soft (digital) goods, such as published material, software, audio and video products.

Business-to-Employee Operation is beginning to develop a new market place. A checkpoint on an emerging application area. As with buying a T-shirt from the company shop; many companies now allow employee to buy using the corporate Intranet. A variant from an emulated business-to-consumer application is where employees may have purchases deducted from the payroll, or from allowances. Allowances or entitlements for clothes or equipment are often the norm in the armed services, police, fire services, airlines, banks, health services and so on.

5. Internet Benefits to Business Operations

To date the major benefits from the Internet include improved internal and external communications. The Web has specifically brought a new marketing medium and enhanced information resource. Innovative applications are starting to appear which allow for sales and database interrogation. Other benefits such as e-mail and file transfer functionality, Web utilization gave many companies *'Internet presence'* and provided them with opportunities to develop and expand new services.

According to the Cisco Systems Inc. (a leading maker of Internet equipment), estimated more than \$1 trillion to \$2 trillion worth of goods and services will be sold on the Net by 2002. According to the Cisco, part of the reason for the low-ball estimates by market analysts is that many exaggerate the importance of business-to-business operations and underestimate the potential growth of the consumer market as the Net becomes more mainstream. On the other hand, Gartner, a leading industry consulting firm, has estimated business-to-business electronic operations will be 12 to 15 times larger than consumer markets for the next few years, with consumer sales only catching up with business markets midway into the next decade.

The Internet can be used to interconnect companies, by-passing any of the standard structures of EDI and VANs. A more significant role for the Internet has been explained by Socka (1996). In Socka's views, the regular EDI structured transactions are treated as a special type of e-mail attachment by the Multipurpose Internet Message Extension (MIME) standard. At the receiving end, the messages are simply printed out without any further translation. Furthermore, the Internet browsers, with their Hypertext Mark-up Language (HTML), are easy to adapt to many types of information exchange compared to EDI standards which are very structured and rigid. In addition, the Internet is very easy to use, while EDI requires trained personnel to operate it.

The Internet offers some impressive possibilities including the Internet transmissions rates, which are cheaper than EDI. According to Bell (1998: 16) the traditional EDI can reduce the cost of ordering to about \$2.50 per order. However, the Internet could reduce that cost even further to less than \$2 per order. In addition, through the Internet, messages can be sent faster compared to EDI, because they are sent directly from one computer to another computer via web routers. Most EDI users today send EDI transmission in overnight batches to save time. EDI usually takes eight to ten hours and often twenty-four hours, while messages through Internet arrive in minutes not hours or days. Pyron, (1996: 84) stated that: “ The speed is actually higher than over a VAN because there is no mailboxing”. Pyron also added: “ We’ll often have people tell us they’ve received an order we’ve just sent while we’re on the phone with them”.

According to Garrison (1998), manufacturers are showing more interest in Internet technology over Electronic Data Interchange. Furthermore, Garrison (1998) stated: “Despite EDI’s active part in manufacturing, suppliers and customers linkage, price quotes and shipping notifications, among other data, Product data change, the medium’s adoption has been limited”. This means, manufacturers are more interested in Internet, because it potentially offers a more feasible means for manufacturers and trading partners to communicate electronically. Garrison (1998) further reported that three-quarters of the respondents to a recent survey said that Internet is a useful communications tool and that only a few manufacturers use EDI often.

Manufacturers embarking on using the Internet initially develop corporate web sites to use as a promotional vehicle, to promote their company’s icon and/or to attain new sales prospects. On the other hand, those manufacturers who are sticking with EDI, found that distributors saw Value Added Network charges as discouraging, and would often fail to check warehouse inventories because of the added communication costs. According to Gallo (1997), “These cost-avoidance habits would lead to inefficiencies, such as ordering from warehouses that no longer stocked a product”. The Internet-based technology provides a cheaper solution to this problem.

There are various many of key measurements that could be used to measure the benefits obtainable from E-Business Operations. Some of these important key measurements are:

- **Improving Customer Service:** Providing customers self-access to their accounts, transactions and orders, is a valuable service. The level of satisfaction for those customers interacting electronically will undoubtedly rise.
- **Reducing Costs:** The most basic cost reductions could be related to publishing costs, which include the cost of production, printing and distribution. Furthermore, marketing and selling costs are also lower in an electronically enabled commerce environment.
- **Providing Business Intelligence:** In the Electronic Business Operations world, businesses need to know much more about their clients. Electronic commerce makes it possible to market to specific individuals based on their patterns of (purchasing and browsing) behavior. Hence they need to capture, and to analyze, as much information as possible about each individual purchase (or cancelled purchase) in order to build up customer's profiles. This is achieved in much the same way that neighborhood stores once did, through personal acquaintance with the consumer and continuous contact. The use of this analyzed data leads to what is being called "market response" systems or "adaptive marketing".
- **Process Simplification:** Instead of using paper, using the World Wide Web (WWW) simplifies and speeds the approval process.
- **Generating New Revenue:** The new Internet-Based Electronic Marketplace generates new revenue by selling new products and services specifically designed for the electronic marketplace. Existing product or services can also be sold on the Internet.
- **Taking Faster Decisions:** By receiving information about competition through an Intranet information retrieval database, it would be possible to develop a competitive strategy much faster than otherwise. The drivers for manufacturing are customer’s needs and time. Time is a major source of competitive advantage and competitive pressures requiring production schedules to be shortened.

According to Iacovou et al. (1995), the benefits obtained from EDI use include both direct and indirect benefits. Direct benefits, such as reduced transaction costs or lower inventory levels, are relatively easy to quantify. On the other hand, indirect benefits such as better customer services and improved trading partner relationships are difficult to quantify. According to Tengende (1993), and Quach (1995), direct benefits take a longer time to eventuate. Furthermore, Swatman (1993) pointed out that to achieve longer-term benefits

from telecommunications-based information systems, an organization needs to combine its inter-organizational systems strategy with existing business strategy.

It is interesting to note that many of the companies interviewed did not state that they had gained direct benefits from the Internet use. In fact, they indicated that the savings and incomes obtained were small. Furthermore, none of the participants suggested that the lack of direct benefits would lead them to cease using the Internet. This suggests that the indirect and non-immediate benefits could be as important (if not more) as direct gains.

6. Success Factors for E-Business Operations

Soliman and Soar (1997) reported that there is a positive relationship between Success Factors and the perception of users of Information systems. This implies that, an efficient way for determining success factors for information systems is through the identification of users' perception of the benefits obtainable from E-Business Operations. Accordingly, this research surveyed the perception of Internet users in a number of manufacturing companies. The pilot survey revealed that users believed that many of the benefits obtainable from using E-Business Operations could include reduced transaction costs; better customer service; more efficient information access; global communications; and shortened communication cycles. In additions users also identified a number of obstacles (impeding) factors. The perception of users were then translated into the following seven success factors for E-Business Operations:

1. ***Security of E-Business Operations:*** Security of conducting of information transmitted through the Internet has been identified as a major success factor. Although, security is currently being improved by available encryption methods, but the biggest security threat is still there according to McCartney (1997). This is emerging as a major deterrent for using the Internet. Accordingly, the better the security of transactions, the more likely manufacturers will use E-Business Operations.
2. ***Support for E-Business Operations and availability of Help Desks and Hot Lines:*** One important issue is the overall network reliability that strikes as a structural feature of the Internet. No one is responsible for its performance. Orr (1996) stated: "If you have any problem, there is no toll free number you can call for customer service". Therefore the availability of help desks and hot lines support is also a success factor for E-Business Operations in manufacturing.
3. ***Speed of transmission using E-Business Operations:*** It is notable that heavy usage of the Internet (*heavy traffic on the Internet*) can slow down the processes and ultimately affect the business usage of the Internet. Accordingly the faster the speed of transmitting information trough the Internet, the more likely manufacturers will use E-Business Operations.
4. ***Sufficient business volume to justify E-Business Operations:*** Many small firms do not see ample benefit from using the Internet. For example, small companies do not indulge in large transactions and may not be engaged in Global Supply Chains. Therefore, sufficient business volume to justify the use of the Internet is also a success factor for E-Business Operations in manufacturing.
5. ***Value Adding of E-Business Operations:*** In general, small companies are nervous and overly cautious when it comes to spending money on technologies that are not absolutely necessary for their operations i.e. Non Value Added technologies. In other words, if the implementation of the Internet results is directly related to the production of goods, then the manufacturers will more likely use the Internet. Therefore Value Adding is also a success factor for E-Business Operations in manufacturing.
6. ***Users Training on using E-Business Operations:*** It has been widely recognised that users' training is an important success factor. In other words, if users are appropriately trained on using E-Business Operations, they are more likely to use it efficiently and effectively. Therefore users' training is also a success factor for E-Business Operations in manufacturing.
7. ***Change management due to the implementation of E-Business Operations:*** The implementation of E-Business Operations will result in inevitable changes. There are three types of Change Management; namely a) Structural Changes, b) Technical Changes and c) Cultural Changes. Managing the three types of changes will reduce fear and would positively impact on use of E-Business Operations in

manufacturing companies. Therefore change management is also a success factor for E-Business Operations in manufacturing.

Yap, et al (1992) and Cragg and King (1993) have identified the level of management involvement as a critical success factor of IT. Soliman and Gide (1998) have also identified management involvement and support as a critical success factor.

Because, different organizations establish different alternative business environments on the Internet, where they can carry out their usual business and networking activities “virtually”, it would be expected that additional success factors applicable to individual organizations should exist.

7. Current Challenges to Internet-Based E-Business Operations

Analysts acknowledge that there are a number of problems facing management in using the Internet to conduct their businesses. Some of these challenges are:

- Providing high quality service to the customer.
- Maintaining service delivery capability, richness of experience and certainty of transaction.
- Generating profits as well as revenues.
- Increasing efficiency and effectiveness of business process.
- Deciding on the level of investment in this technology.
- What to do with existing Electronic Data Interchange (EDI), and when to shift away from it?
- How to use the Internet to create competitive advantage for tomorrow?
- How to focus on delivering solutions to the Internet commerce markets' space?
- How to leverage IT leadership to drive growth and direction of the Internet market?
- How the Internet leads to best business practices?
- How to deliver a viable, cost effective and compelling Electronic Business solution?
- How to expand into new markets?

The phenomenal predictions of the size of the Internet market should be interpreted with some other factors in mind. There is scant evidence that the Internet is actually creating new sales. Certainly, the Internet is beginning to generate new sales channels, especially for products and services, which can be delivered digitally over the net. And there is no doubt that bank-assumed risk from credit card transactions through SET processes will accelerate traditional retail sales over the Internet. But these sales are still generally no more than sales substitution, or sales that would previously have been made by personal visits, mail order or the like.

There are two main drawbacks or challenges in using Internet-Based Electronic Business Operations, these are: security issues and payment tools. These two issues are receiving the highest priority and the best attention they deserve, both from vendors and users and implementers.

The cost-benefit justification of Internet access and Information Systems in general is, and always be, a difficult one to prove but due to access of real time data, this would provide long term benefit of immeasurable value. To be able to analyze the value of Internet Commerce usage, it is helpful to have a measuring ruler. For individuals, one can compare the cost of an Internet connection to the cost of using the telephone. It has been demonstrated that e-mail is cheaper than phone to communicate long distance with a number of people. The Internet Commerce usage can make it possible to reduce the amount of time or effort required to perform certain tasks: cost savings and benefits from providing sales and customer support online and increase the potential of collaborative partnerships established over the Internet.

7.1 Security and Privacy Issues

While EDI users enjoy a high level of reliability and security, they are often restricted to the exchange of data with users of the same Value Added Network (VAN). For electronic commerce to really transform the way we do business a secure solution that works globally is required. To achieve this, a series of international standards needs to be agreed and vendors need to carry out a rigorous program of interoperability tests. Moreover, as trade moves beyond national boundaries, a common legal infrastructure

must be agreed. For example, a contract that has been digitally signed in one country needs to be recognized in other countries.

On the other hand, a recent Forrester Research report found that security has fallen from first place in 1995 to fifth place in 1996. This indicates that there is a growing confidence in solving the Internet security issues that have been very widely publicized. Even though security is a challenge it is not a barrier to Electronic Business Operations.

Security is fairly new to the Internet, so it has not matured yet. However, computer security professionals have known about the Internet security for years and are now improving it.

7.2 Payment Tools

There is confusion over the availability and choice of Internet payments tools. In addition, there are no interoperability standards to make one work with another. Over the past two years, new payment tools from small companies have emerged.

There are many traditional methods of payment available in the real world such as: Cash, Cheques, Credit Cards, Traveller's Cheques, Prepaid Cards, Debit Cards, Physical Tokens, Bank Notes, Secure Wire Transfers, Money Orders, Letters of Credit, etc. However, none of these mechanisms is directly transferable in an unmodified form to suit the Internet. This is because each method assumes a physical presence or that there is a delay incurred in the processing of funds so that fraud can be detected and stopped. Some of the new Electronic Business Operations payment tools that can be used in business operations are:

1. **Electronic Cash (Digital Cash)**- It is a token-based currency which translates into equivalent real currency units that are guaranteed by a bank. Usually, there is a trusted authority that allows the user to conduct and pay for transactions of this nature. This usually takes place after a pre-determined relationship has been established (e.g. DigiCash).
2. **Smart Cards**- Smart Cards can be used with or without a stored value. Usually, the user is able to pay with them without having to connect to a remote system. If they have a stored value which contains "real digital cash", they are known as "Cash Cards" because they replace carrying cash (e.g. Mondex).
3. **Electronic Cheques**- These are the equivalent of paper based cheques. They are initiated during an on-screen dialog which results in the payment transaction. Authentication and verification are usually performed instantaneously by using digital signatures and time-stamping controls during the transaction (e.g. CheckFree).
4. **Encrypted Credit Cards**- There are varying degrees of encryption implementations credit of credit cards over the Internet, with the SET (Secure Electronic Transactions) holding the most promise (e.g. CyberCash).

8. Barriers to Implementing E-Business:

Some experts admit that the Internet gives a platform where small companies can do business with one another even with large suppliers. But they do not expect large corporations to abandon their well-structured and planned EDI networks in the near future. This is largely due to 3 main reasons, which are:

- ⇒ **Security on the Internet:** Security on the Internet is a major problem. This is currently being handled by available encryption methods, but the biggest security threat is still there according to McCartney (1997).
- ⇒ **No one owns the Internet:** One important issue is the overall network reliability that strikes as a structural feature of the Internet. No one is responsible for its performance. Orr (1996) stated: "If you have any problem, there is no toll free number you can call for customer service".
- ⇒ **Heavy traffic may delay the business operations:** It is also notable that heavy usage of the Internet can slow down the processes and ultimately affect the business.
- ⇒ **Method of Payment:** There is no universally accepted method of payment over the Internet.

9. Model For E-Business Implementation:

Soliman (1999) has shown that the introduction of the Internet in organizations is a very complex and is an evolutionary process and that organizations embarking on using the Internet to conduct their business

operations or part of them will soon discover that they have to pass through the following three distinct but interrelated stages:

1. Searching for Business Opportunities (The Interactive Marketing Stage).
2. Communicating with suppliers and customers (The E-Commerce Stage).
3. Conducting full-scale business transactions (The E-Business Stage).

A model for describing the three phases of development and introduction of E-Business has been provided by Soliman (1999) and is illustrated below in Figure 3 below.

The model shows that as the relationship between Business-Business, Business-Customers and Business-Employees matures, the pattern of information exchange becomes more structured and intensive. The model is useful in analyzing how organizations use the Internet to supplement their typical business activities and how much it is used as E-Business.

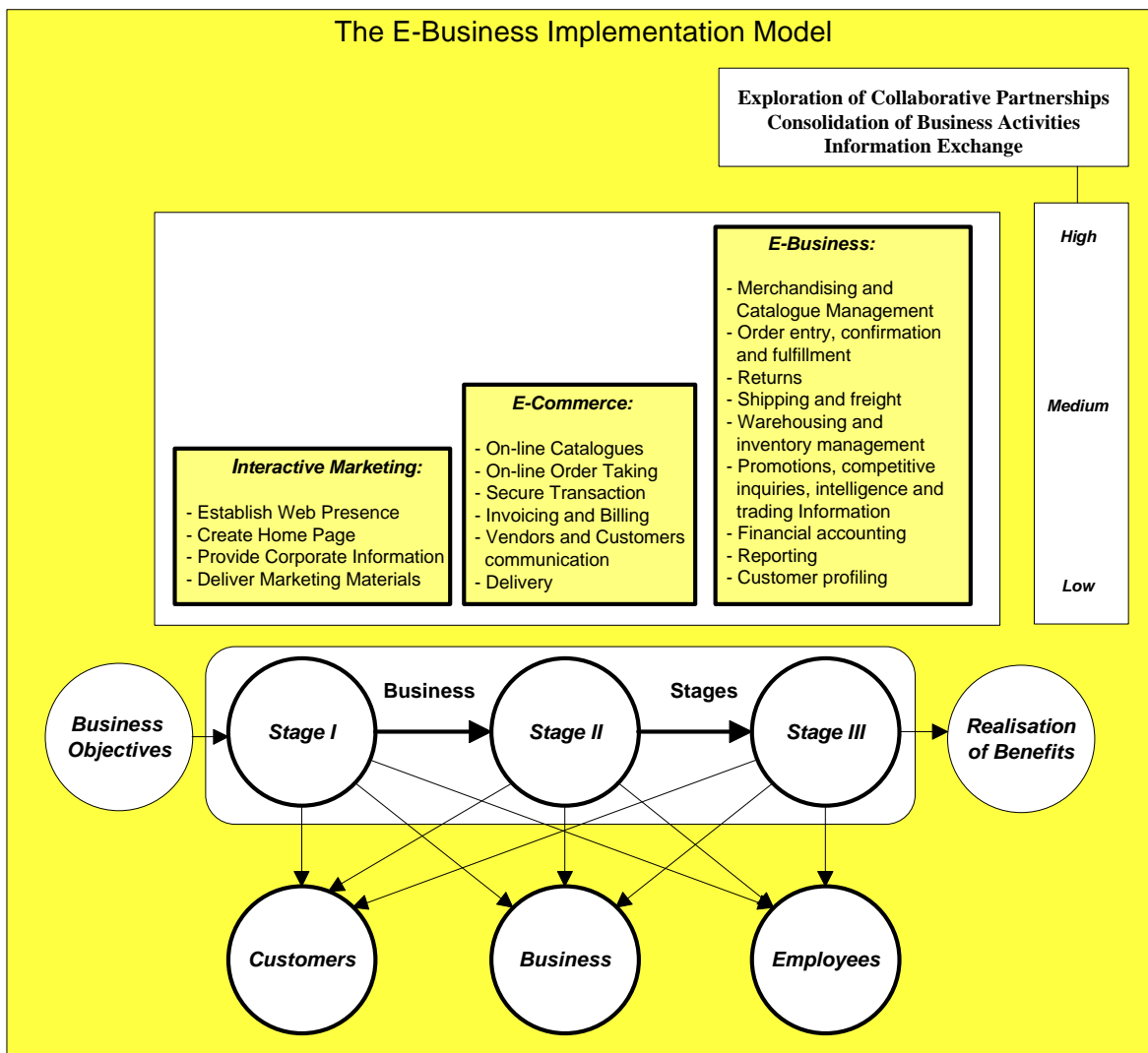


Figure 3: Model for the development and introduction of E-Business

The model also illustrates that the three phases of E-Business introduction are distinctively different in their business focus as explained below:

Stage I: Searching for Business Opportunities

The Internet, and in particular mailing lists and Usenet newsgroups, can be used to explore potential business opportunities. This is the *Web Presence Stage* and is the most elementary stage. During this stage the user is mainly concerned with delivering to potential clients and customers marketing and promotional material. The process of Searching for Business Opportunities can be either a one- or two-way approach. For example, the company which has a Web page to advertise and market its products can do so through Usenet newsgroups (such as alt.industrial). The company responds to enquires and at the same time provides product information. The main activities at this stage are:

- Establish web presence,
- Create home page,
- Provide corporate information, and
- Deliver marketing materials.

Stage II: The E-Commerce

Once the first stage achieves its aims and the company is still willing to pursue the use of the Internet to carry out some of its operations, the second stage the *Electronic Commerce Stage* begins. The Electronic Commerce stage enables companies to conduct some of their usual operations on the Internet. Following are some of the activities that can be conducted during the Electronic Commerce stage:

- Creating online catalogues;
- Online order taking and order placing;
- Communicating with suppliers;
- Conducting banking and making payments;
- Securing transactions; and
- Making delivery.

Stage III: The E-Business

The third stage the *Electronic Business Stage* commences when the company decides to embark on full-scale business activities on the Internet. The most important business functions that are likely to be conducted during this stage are:

- Merchandising and catalogue management;
- Order entry, confirmation and fulfillment;
- Returns;
- Shipping and freight;
- Warehousing and inventory management;
- Pricing - promotions, taxes, duties and freight;
- Payment cycles-credit cards, digital cash, bank transfers;
- Financial accounting;
- Reporting; and
- Customer profiling.

The model also shows the introduction of the Internet result in the emergence and strengthening of following three important E-Business Operations:

9.1 E-Business Operation 1: Exploration of Collaborative Partnerships

During each stage, the Internet is used as a medium to explore collaboration possibilities, without incurring heavy time or expense overheads. This is achieved through enquires posted on a mailing list and a number of Usenet newsgroups. This enables the manufacturer to make contact with a user who has published on this particular issue. Although the Internet is a very useful tool to explore collaboration possibilities, further consolidation of the initial business contacts requires more direct interaction or perhaps another communication medium.

9.2 E-Business Operation 2: Consolidation of Business Activities

After identifying collaboration activities and pinning down preliminary collaboration plans, most participants feel comfortable, at least partially, in resorting to electronic mail for further exchanges. By now, the collaborators have already established a more in-depth business relationship and they know the people with whom they are communicating. What seems to be important is the 'handshake', which makes further virtual communications worthwhile. All participants agreed that Internet-based communication is preferred if it is more convenient and effective to both parties involved in the communication. At the same time, all agreed that telephone, fax, post and most often face-to-face meetings are still necessary. Electronic mail and the Web had been used during the various stages of business development.

9.3 E-Business Operation 3: Information Exchange

All users tend to use the Internet fairly extensively during all stages of business development. Again, electronic mail is the principal service used to exchange documents and information. It appears from the variety of activities that carried out on the Internet, the Internet seems to be most useful for opportunity search and routine message exchange. However, when deeper mutual understanding is needed, richer communication media such as teleconferencing or even face-to-face interaction are important.

Information exchange increases the ability to seize business opportunities quickly and exploit them to the manufacturer's full advantage so that future competitiveness is consolidated. This is similar to the ability to perceive the market potential of a new product and carry out rapid product development to capture market share. Both the ability to conceive bright ideas and to make them work in practice, will transform intangible benefits to tangible outcomes in Internet use. Furthermore, it appears that the manufacturer can gain and preserve their edge by continuously applying Entrepreneurship in business use of the Internet.

10. The Strategic importance of E-Business Operations

For gaining strategic advantage through Information Technology managers must understand not just the technology but also the "value chain" in which their company operates. Information technology and in particular Internet E-Business will have an impact on each activity along the value chain. In fact, E-Business Operations are transforming the way value activities are performed and the nature of the linkages among them. These basic effects explain why information technology (Internet) has acquired strategic significance and is different from the many other technological business use.

The Internet can make a significant contribution to each components of a company's value chain (Mougayar, 1997). To uncover and evaluate new avenues for competitive advantage through use of the Internet, companies need to analyze their relationships with suppliers and vendors, the existing role of information in the organization of the company, internal production mechanisms, and the points of contact with customers.

The Internet can make a significant contribution to each components of a company's value chain (Cronin, 1994). To uncover and evaluate new avenues for competitive advantage through use of E-Business Operations, companies need to analyze their relationships with suppliers and vendors, the existing role of information in the organization of the company, internal production mechanisms, and the points of contact with customers. Management via Internet-Based E-Business Operations could improve the competitive advantage is shown below in Figure 4.

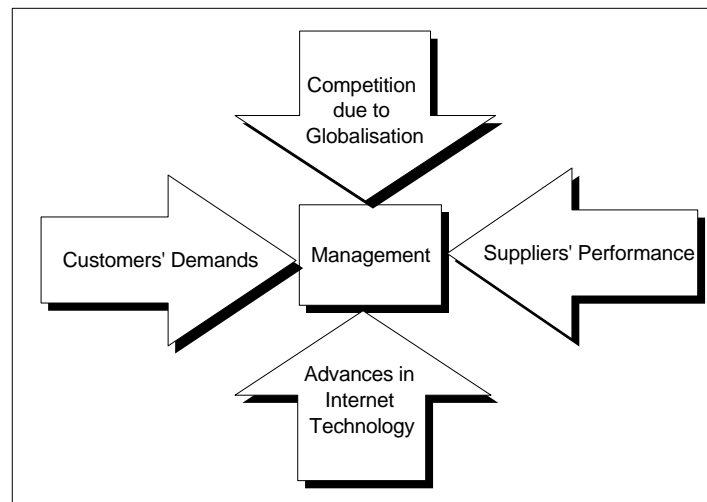


Figure 4: Management via Internet-based E-Business to improve the business Competitive Advantage.

The above figure indicate that management of modern organizations will come under pressure from four different but interrelated fronts:

1. *Customers' Demands*: Customers demand, better quality, faster response and cheaper goods and services.
2. *Suppliers' Performance*: Suppliers' performance is crucial to organization's ability to meet its customers' demands. In addition suppliers demand flexibility, faster and timely response and faster method of payment.
3. *Competition due to Globalization*: Globalization has resulted in intense competition. This requires flexibility, productivity, and ability to make fast decisions using accurate, timely and up-to-date information.
4. *Advances in Internet Technology*: To meet its strategic objectives, an organization needs to implement E-Business Operations and systems which are compatible with those used by its suppliers, customers and stakeholders. These systems are in fact Strategic Systems. (Soliman, 1999)

Clearly, the Internet provides very fast, reliable connections to suppliers, and customers around the world. Using E-Business Operations companies can communicate with vendors in any location, without incurring additional communication costs. Sometimes even overnight delivery of information may be too slow when critical decisions are waiting to be made. Many vendors offer electronic pricing and ordering information to overcome these limitations.

The availability of electronic distribution software, publications, and other items provides immediate access to many products and makes On-line tracking of orders and inventory are viable E-Business Operations. This ensures that companies are aware of delivery dates, and reduces delays in the distribution process. Many companies have found that product support over the Internet significantly reduces the time lost due to system performance problems (Cronin 1996a). For some companies, the efficiencies and cost savings generated by dealing directly with suppliers over the Internet have more than justified their investment in the network.

The global connectivity of the Internet offers companies immediate savings in long-distance telecommunications. A dedicated Internet connection allows unlimited exchange of data and e-mail with locations around the world. Even a low-cost, shared dial-up connection with an hourly use charge is more economical than long-distance telephone charges. In the longer term, the ability to exchange information quickly and easily facilitates the relationships with business partners and customers, encouraging more joint ventures. For employees, connecting to an international information source promotes global awareness. It allows companies to monitor economic and political developments in countries targeted for market expansion.

The Internet allows direct interactions with customers to be spread through many divisions of a company; technical and development staff, documentation providers, production workers, and researchers find out first hand how customers respond to company products.

In addition to that, strategic use of the Internet based on an analysis of the value chain encourages companies to focus on areas where they can measurably improve performance. The benefits of the Internet will vary from business to business. One thing is certain that for companies seeking competitive advantage, the global network is essential management resource.

11. The Strategic use of E-business Operations

In this rapidly changing environment, businesses are taking a look at their own organizations, structures, and processes in an effort to become more competitive. Many companies are using e-mail and group conferencing to engage in business process re-engineering projects. Maintaining good communication to exchange data and documents is critical in the re-engineering of business processes. The competitive benefits of the Internet E-Business Operations are illustrated below in Figure 5.

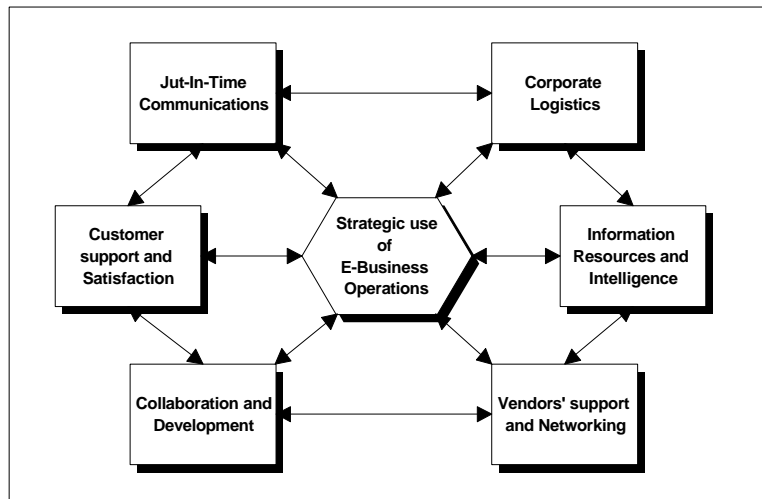


Figure 5: The strategic use of the Internet E-Business Operations

In addition, the ability to have the latest information about the marketplace and awareness of the state-of-the-art in the industry allows a company to keep its competitive edge. Learning what other companies are doing, knowing the kinds of information available, and discovering new markets can assist a company in maintaining a competitive advantage.

More companies use the Internet in the search for “best practices.” As businesses try to become more competitive, many want to find existing practices that can help them improve their activities. Businesses can also use the communications abilities of the Internet to engage in a Total Quality Management (TQM) plan. Some companies use the Internet to maintain corporate process control across all company locations including even continents. Many companies use the Internet commerce tools to search for successful practices of corporate and product improvement.

Competitive intelligence can be gained due to access to state-of-the-art information on products, materials, new ideas and even the status quo in a given industry. Companies can use the Internet to engage “techno-watch”- keeping a finger on the pulse of emerging and new technologies, and the market response to those technologies, both anecdotal and in terms of financial performance and the stock market. The main strategic benefits of using the Internet-based E-Business in operations are briefly summarized as below:

11.1 Just-In-Time Communications

The Internet offers a business the opportunity for just in time (JIT) communications with people and organizations across the globe. Being on the Internet allows a company to truly have a world market. Good communications enable more global corporate management control, aiding in consistency of results. Companies can be in touch with suppliers, branches, and subsidiaries in an effort to exert more control over variables. Companies can establish, negotiate, and maintain standards on-line and find out: What are other businesses doing? What kinds of information are available? Who are the main competitors in a specific business?

11.2 Corporate Logistics

Logistical concerns can dominate planning and customer service in many corporations. Since the Internet is the anywhere-anytime-network, employees, suppliers, customers, and stakeholders can keep in touch more efficiently. The use of e-mail and teleconferencing facilitates communication between markets. In addition, Just-In-Time communication is also possible. Using the Internet for communication removes distance and time barriers.

11.3 Collaboration and Development

The development team and project participants often use the Internet to keep in touch, and to exchange data, programs, and working papers from any locations. The Internet also allows several small businesses to band together much more easily for product development. In another words, formation of partnerships among companies is increasingly common, and the Internet facilitates the collaboration for product or service design, vendor channels, research, and development. Collaborative approaches can be greatly enhanced by the Internet with its wealth of information, its capacity of supporting telecommuting and time-shifted communication.

11.4 Customer Satisfaction Through Support

In this highly competitive and dynamic global marketplace, the company that can reach and satisfy customers will have an advantage- and the Internet E-Business Operations can help in maintaining relationships with customers. With its global reach, the Internet can assist business in locating new suppliers and keeping in better touch with them to aid, for example, in lower sometimes even zero inventory planning.

One of the main business uses of the Internet E-Business Operations is in the area of customer support. Customers can reach a company's homepage on their own schedules any time and be able to obtain information regardless where they are. Many companies maintain World Wide Web sites, Gophers, and FTP sites for customers use during working and non-working hours. These services enable customers to receive assistance, get product information, and leave questions for replies during working hours and offer information files on Frequently Asked Questions (FAQs) for customers or potential customers.

11.5 Vendor Support and Networking

The Internet provides a fast method for networking with vendors and suppliers, increasing speed and variety in procurement process. The Internet can help businesses locate new suppliers and keeping better touch with them. In addition, small suppliers can network with and compete with larger, better-known suppliers. Furthermore, the Internet assists companies in maintaining low inventory levels because of speed of communications. Relationships with vendors and outlets can also be maintained via the Internet. Companies can do actual sales transactions on the Internet. Companies can arrange product delivery through the Internet, where companies can establish and support actual distribution channels.

11.6 Information Resources and Competitive Intelligence

With more than 20 million (1998) machines connected to the Internet, the system has a multitude of databases, Web sites, Usenet, Gopher sites, FTP (File Transfer Protocol) sites, Listserver discussion lists, and conferences, the amount information available is staggering. Scientific and research data is available in large quantities. Furthermore, some companies find that the Internet is useful in helping employees learn new tasks and processes. There are many simulations, manuals, training aids, and tools available for software running on a variety of platforms, from UNIX tutorials to Windows tips and hints. There are also large quantities of instructional materials available on-line regarding the use of the Internet.

12. Conclusions

Business Operations over the Internet is very much in the early stages. Early indications are that the-Internet is a viable trading medium. The problems of cost, standards and a lack of interactivity will prohibit traditional batch-EDI scenarios. Success in Internet-Based Electronic Business Operations depends on how organizations strategically position their products and services through other Internet-based electronic

communities and intermediaries, as well as on how they facilitate the interactions with their customers, suppliers, and partners. Even though Electronic Business Operations make sense *theoretically*, the reality is that it has to integrate with internal and external processes that are already in place. Sometimes, this integration is a challenge linked to a major re-engineering exercise accompanied by resistance to change. Moreover, since implementation of Electronic Business Operations is, in many cases, evolutionary, organizations need to react to change the business processes as demand increases.

On the other hand, as discussed early the Internet Commerce is one of the most important strategic tools for manufacturing management. Businesses of all types and sizes can also find that the Internet serve a large variety of their needs as a strategic tool to gain competitive advantage over their rivals, including marketing, customer and vendor support, the exchange of information, and joint ventures for research and development.

With the aid of the Internet-based E-Commerce, companies also can develop new products, communicate in real time, take orders, receive electronic publications and documents, and retrieve data from specialty databases. Businesses can find technical advice, establish and maintain business relationships, obtain market intelligence, ferret out good deals, locate people with needed skills, and even provide products directly.

Despite these benefits and success stories, a number of issues remain to be resolved such as security, privacy and payment tools. Other issues regarding the growth of the Internet are the lack of: a public key infrastructure (particularly for international trade), governmental stance, access, reliability (service levels), integrated applications and understanding/awareness of the Internet-Based Electronic Business Operations capabilities, and finally the relative cost of required technologies.

The results of this study suggest that perceived benefits and management involvement are two factors common for all the participating companies with a positive effect on E-Business use. By reviewing the way in which the Internet is used during the different stages of a business relationship, it would be possible to construct a model for manufacturers' use of E-Business. The limitations of the results generated by this study are partly due to the small sample size used.

By interviewing a group of three E-Business users from a variety of manufacturing sectors, it was possible to discover that while competitive advantage does appear possible, it can as yet be described only as a "perceived" benefit. The results of the interviews were a key factor in constructing the model that reflects the way in which the E-Business has been used during the different stages of business development.

Finally, this paper outlined the research method used and provided a summarized background of E-Business and the case study used. Perceived benefits of the Internet and management involvement in adopting the Internet for business use appear to be key driving forces for continued Internet use. With the aid of the developed model and the case study results, it was possible to analyze the ways in which the Internet is used as a supplement to the traditional business environment.

13. Recommendations:

It is suggested that further research needs to be carried out to extend the in-depth study of factors such as entrepreneurial approaches to E-Business use among small businesses. Examination of the importance of external factors, such as business culture and technology availability, during the implementation of E-Business is also recommended.

References

1. ActiveMedia (1996), www.activemedia.com
2. Bell, S. in Sliwa, C. (1998), "Internet used to extend EDI's reach", *Computer world*, Vol. 32(8), pp. 1-16.
3. Cisco Systems Inc. (1998), www.cisco.com
4. CommerceNet (1998), www.commerce.net
5. Cronin, M. J. (1994), *Doing Business on the Internet: How the Electronic Highway is Transforming American Companies*, Van Nostrand Reinhold, New York.
6. Cronin, M. J., (1996a), *Global Advantage on the Internet*, Van Nostrand Reinhold, USA
7. Cronin, M. J., (1996b), *The Internet Strategy Handbook: Lessons from the New Frontier of Business*, Harvard Business Press, USA.

8. Cragg, P. B. and King, M. (1993), "Small-Firm Computing: Motivators and Inhibitors", *MIS Quarterly*, March, pp. 47-60.
9. Forrester Research's Business Trade & Technology Strategies Service, (<http://www.internetnews.com/ec-news/cur/1997/07/3005-bb.html>)
10. Gallo, J in Abcede, A. (1997), " EDI, Internet Connects as data goes electronic", *National Petroleum News*, Vol. 89(11), pp. 110-114.
11. Garrison, S. (1998), "Survey: Extranets could deliver what EDI didn't", *Systems Management*, Vol. 26(2), p. 34.
12. Gartner (1998), www.gartner.com.
13. Gide, E. and Soliman, F. (1999), "Framework for E-Commerce Implementation in Business Operations", *Proceedings of the 2nd International Conference on Innovation Through Electronic Commerce*, Manchester (UK), November 1-3, 1999, In press.
14. Gide, E. and Soliman, F. (1998): "Framework for the Internet-Based E-Commerce in Manufacturing and Business Operations", *Proceedings of the "NETIES'98: Networking for the Millennium"*, Leeds, 15 – 16 October, pp 66-72.
15. Gide, E., and Soliman, F. (1998), "A Model for the Implementation of E-Commerce Over the Internet in *in the Proceedings of IMS'98 International Conference*, Sakarya, 6-7 August 1998, pp. 801-811.
16. Gide, E., and Soliman, F., (1997a): "Analysis of Conducting Business on the Internet, in the *Proceedings of Inet-tr'97 Conference*, Ankara, 21-23 November 1997.
17. Gide, E., and Soliman, F., (1997b): "Key Drivers for Using the Internet in Australia," in the *Proceedings of Inet-tr'97 Conference*, Ankara, 21-23 November 1997.
18. Hoffman, D. and Novak, T. (1996), "A New Marketing Paradigm for Electronic Commerce", URL: <http://www2000.ogsm.vanderbilt.edu/novak/new.marketing.paradigm.html>.
19. Paul Kagan and Associates (1997), www.paulkagan.com
20. Iacovou, C. L., Benbasat, I. and Dexter, A. S. (1995), "Electronic Data Interchange and Small Organizations: Adoption and Impact of Technology", *MIS Quarterly*, December, pp. 465-485.
21. Kalakota, R., and Whinston, A. B., (1996), *Frontiers of Electronic Commerce*, Addison Wesley, USA.
22. McCartney, L. (1997), "A safety net", *Industry week*, Vol. 246(8), pp. 74-78.
23. Mougayar, W.,(1997), *Opening Digital Markets*, CyberManagement, Canada.
24. Nielsen Media Research (1998), www.nielsen.com
25. Orr, B. (1996), "Will the Internet get EDI going", *ABA Banking Journal*, Vol. 88(2), p. 70.
26. Poon, S. and Swatman, P. (1995), "The Internet for Small Businesses: an enabling infrastructure for competitiveness", *Proceedings of the Fifth Internet Society Conference*, Hawaii, June, pp. 221-231.
27. Pyron, G., in Ubois, J. (1996), "Net dreams for EDI", *The Magazine for Chief Financial Officers*, Vol. 12(11), pp. 83 - 85.
28. Quach L.C. (1995), "Cost-Benefit Analysis of EDI: A Means for Justification", Department of Information Systems Honours thesis, Monash University, Melbourne, Australia.
29. Rayport, J. F. and Sviokla, J. J. (1995), "Exploiting the Virtual Value Chain", *Harvard Business Review*, Nov-Dec, pp. 75-85.
30. Socka, G. (1996), "EDI meets the Internet", *Cost and Management*, Vol. 70(5), pp. 14-17.
31. Soliman, F. (1999), "Success Factors for Implementation of Internet-Based E-Business", *Proceedings of the 1999 International Wireless and Telecommunications Symposium*, 17-21 May, Shah Alam, Malaysia: 354-359.
32. Soliman, F. (1998): "A Model for the Introduction of E-Business", *Proceedings of the "NETIES'98: Networking for the Millennium"*, Leeds, 15 – 16 October, pp 55-59.
33. Soliman, F., and Gide, E., (1997): "Impact of Internet-based E-Commerce on Manufacturing and Business Operations," in the *Proceedings of Inet-tr'97 Conference*, Ankara, 21-23 November.
34. Soliman, F. and Soar, J., (1997), "Physician Clinical Communication Systems - An Australian Perspective, *Journal of Medical Systems*, Vol. 21, No. 2: pp 99-106, April 1997.
35. Swatman P. (1993), "Integrating Electronic Data Interchange with Existing Organizational Structure and Internal Application Systems: the Australian Experience", PhD Thesis, Curtin University of Technology, Perth, Western Australia.
36. Tengende P.K. (1993), "Developing a Metric for Cost Benefit Analysis of EDI Projects", Master of Science (Computing) thesis, Curtin University of Technology, Perth, Western Australia.
37. Yap, C. S., Soh, C. P. P. and Raman, K. S. (1992), "Information Systems Success Factors in Small *International Journal of Management Sciences*, Vol. 20(5-6), pp. 597-609.

Managing the E-Business Operations

Quality Week Europe 1999!

QWE'99

November 1-5, 1999, Brussels, Belgium

Dr Fawzy Soliman

School of Management

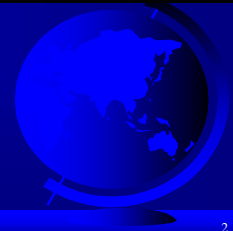
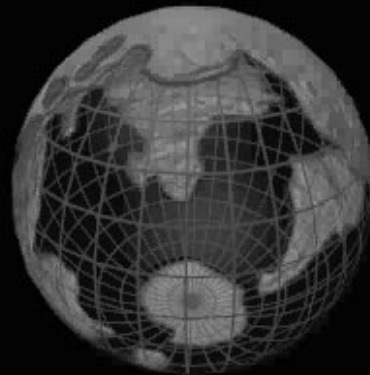
University of Technology, Sydney, Australia

E-mail: Fawzy.Soliman@uts.edu.au

© Dr Fawzy Soliman QWE'99

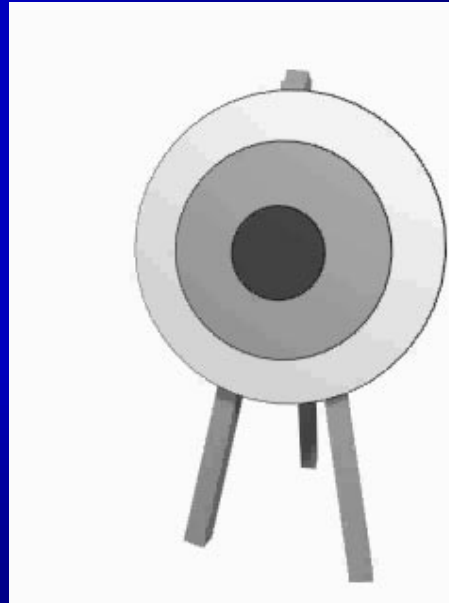
Agenda

- ❖ Objectives of Research.
- ❖ Definitions (Examples).
- ❖ Future business outlook.
- ❖ Role of Internet E-Business.
- ❖ Benefits.
- ❖ Obstacles.
- ❖ Driving Forces for E-Business.
- ❖ Model For E-Business Implementation.
- ❖ Research Problem And Methodology.
- ❖ Results (Types, Challenges and Stages).
- ❖ Conclusions.



Aims

- ❖ Study how organisations conduct business on the Internet.
- ❖ Present a model of how organisations develop and implement Internet E-Business.
- ❖ Study benefits and potential business opportunities for companies from Internet E-Business.

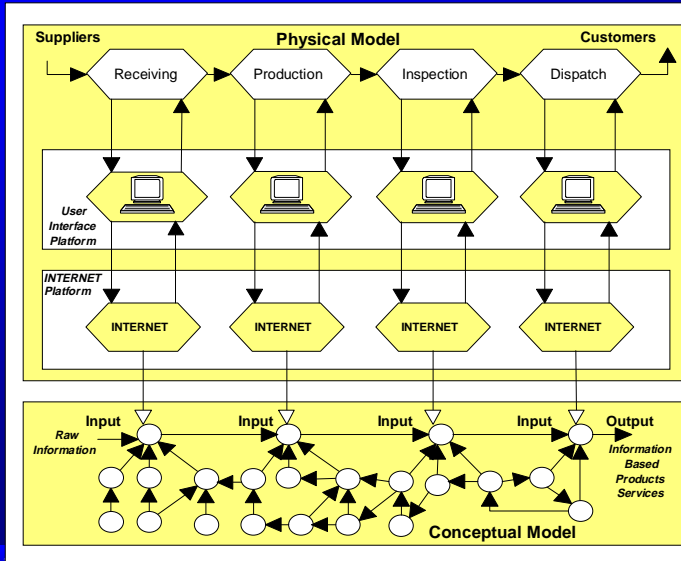


What is Internet E-Business?

- ❖ Electronic Commerce!
- ❖ Electronic Trading!
- ❖ Electronic Data Interchange (EDI).
- ❖ Computer-to-Computer transactions.
- ❖ Major transformations of firms.
- ❖ No exact definition?



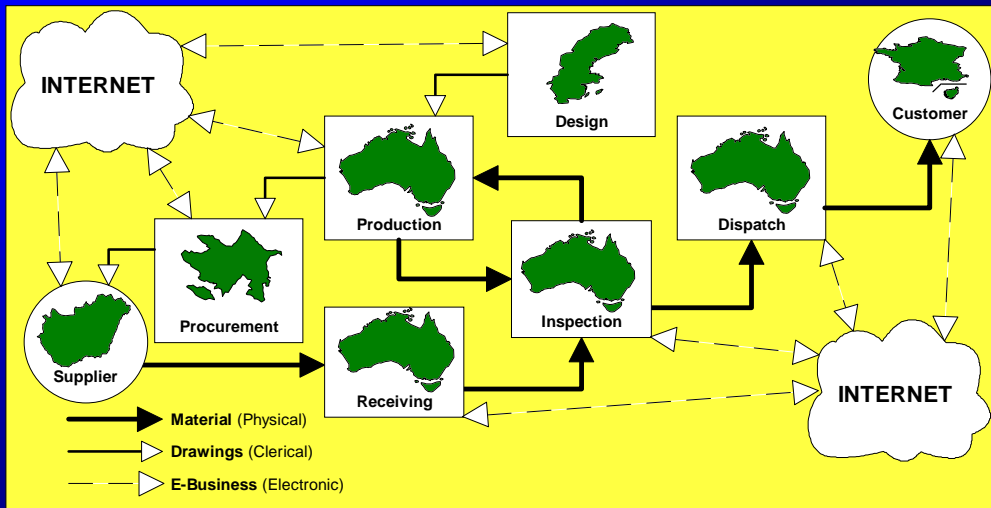
Process of E-Business



Kalakota (1996) "the process of converting digital inputs into value-added outputs".

Supply Chain?

Supply Chain using the Internet



Future Business Outlook?

Future Business Outlook

- ❖ Dynamic and turbulent business environment.
- ❖ Globalisation of business operations.
- ❖ Process Focused, BPR and Down Sizing.
- ❖ Merging and consolidation of markets.
- ❖ Fragmented consumer markets.
- ❖ Rapidly changing technologies.
- ❖ Pressure from customers.
- ❖ Fierce competition.
- ❖ Business Intelligence.



Internet?

- ❖ The Internet is a key part of Information infrastructure.
- ❖ Recent survey of US small business using the Internet.

| | Small Firms Overall (%) | Importers (%) | Exporters (%) |
|----------------------------------|-------------------------|---------------|---------------|
| Use of the Internet | 23 | 35 | 39 |
| Conduct research on the Internet | 17 | 21 | 26 |
| Host/have a Web page | 8 | 14 | 17 |

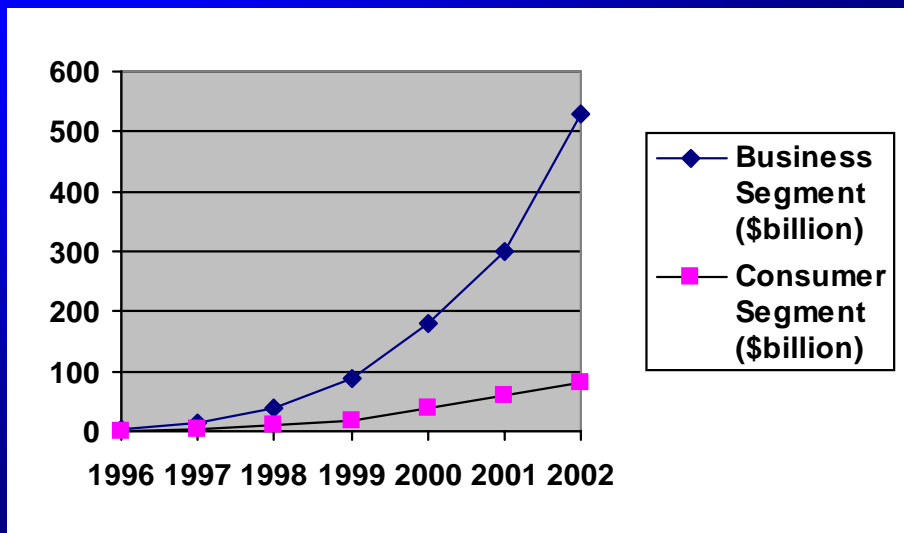
| Reason for Using Internet Commerce | Percentage (%) |
|------------------------------------|----------------|
| Cost Savings | 35 |
| Customer Service | 32 |
| Revenue Generation | 18 |
| Marketing | 13 |
| Others | 2 |

Usage of the Internet for Business

| Industry Type | 1997 % Sales | 1997 \$Billion |
|--|--------------|----------------|
| Manufacturers of electronics and aeroplane parts | 37.5% | 3 |
| Vendors of computer-related and office supplies | 37.5% | 3 |
| Services and utilities providers | 25% | 2 |

| Items Purchased | June, 1998 (million people) | September, 1997 (million people) |
|---|--------------------------------|-------------------------------------|
| Books | 5.6 | 2.3 |
| Computer Hardware | 4.4 | 2.0 |
| Computer Software | 4.0 | 2.8 |
| Travel (airline tickets, hotel & car reservations) | 2.8 | 1.2 |
| Clothing | 2.7 | 0.9 |

Internet E-Business Revenues



Benefits?

- ◆ Benefits can be *direct* and *indirect* benefits.
- ◆ Direct benefits (relatively easy to quantify):
 - reduced transaction costs;
 - reduced cycle time and
 - lower inventory levels.
- ◆ Indirect benefits (difficult to quantify):
 - better customer services; and
 - improved trading partner relationships.
 - make faster decisions.



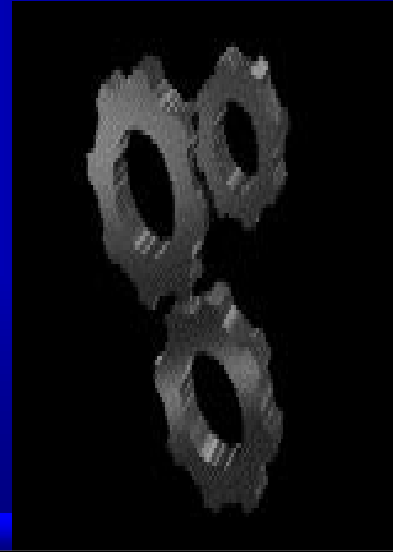
Obstacles

- ◆ Lack of knowledge on:
 - Benefits from Internet E-Business.
 - How to implement and Integrate?
- ◆ Lack of:
 - Financial justification.
 - Value Added analysis.
 - Support from Management, and Employees.
- ◆ Inappropriateness of Internet in an industry.
- ◆ Extensive IT/IS changes or BPR required.
- ◆ Other factors:
 - ⇒ *Security on the Internet.*
 - ⇒ *Method of payment.*
 - ⇒ *No one owns the Internet.*
 - ⇒ *Heavy traffic on Internet may delay business operations and can slow down processes.*
- Φ Small companies: Economic Injustice.



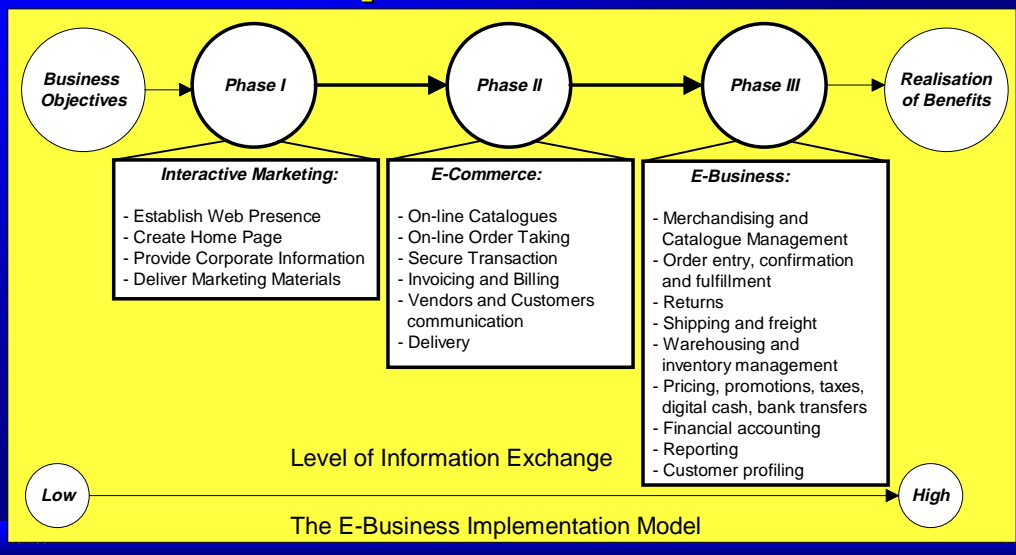
Driving Forces for Internet E-Business Implementation

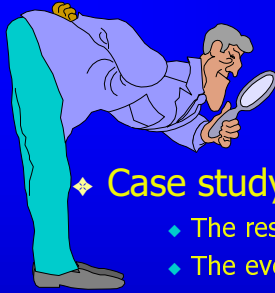
- ◆ Pressure from customers, suppliers, globalisation and competitors.
- ◆ Need to capture more market share (globally).
- ◆ Need to improve firm's image.
- ◆ Need to reduce manufacturing cost.
- ◆ Need to increase productivity.
- ◆ Achieve competitive advantage.
- ◆ Request from Board of Directors.



How to implement?

Model for E-Business Implementation





Research Problem and Methodology

◆ Case study approach because:

- ◆ The researcher has little control over the environment;
- ◆ The events under investigation are contemporary; and
- ◆ The context of the research is important.

◆ Research problem:

"Why and how do Australian companies use the Internet for competitive purposes?"

- ◆ Companies, size of company, views on competitive advantage of Internet and business sector.
- ◆ Interviews and site visits or telephone interviews.



◆ Exploration Of Collaborative Partnerships. **Results**

◆ Consolidation Of Business Activities.

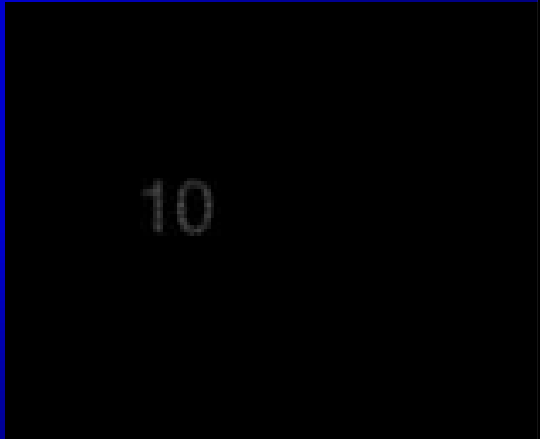
◆ Information Exchange.

◆ Three types of Businesses:

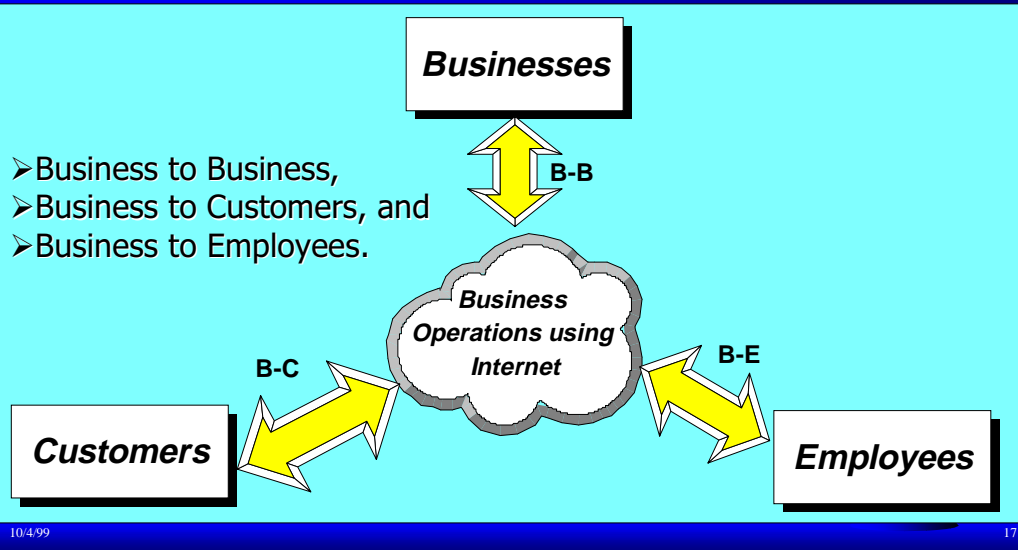
- Business to Business,
- Business to Customers, and
- Business to Employees.

◆ Critical Success factors:

- IT Infrastructure.
- User training.
- Security.
- User Appreciation (Involvement and satisfaction).
- IT Department's Support for Internet implementation.
- Management commitment and Support.

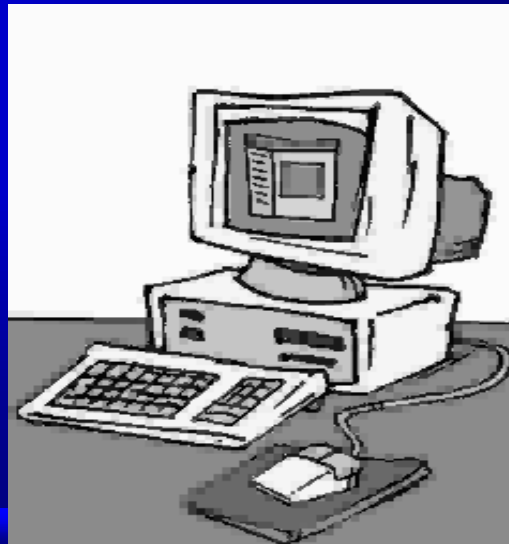


Results: Types of E-Business Operations

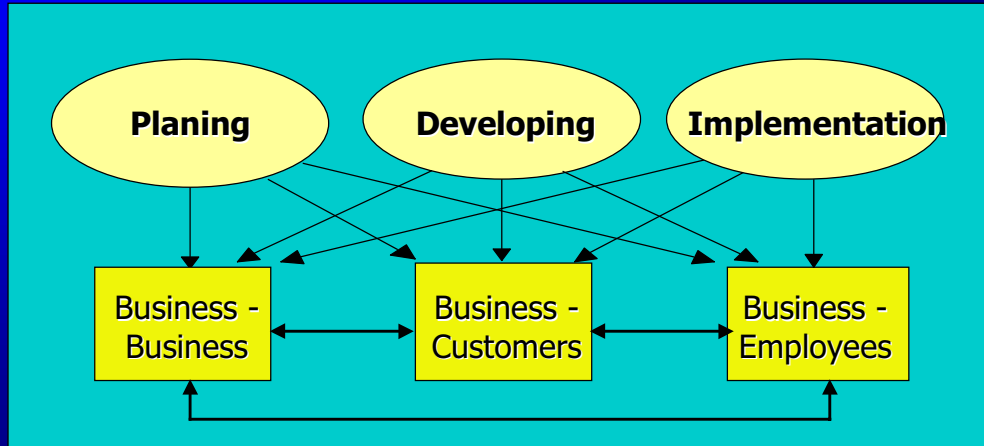


Results: Challenges To Internet E-Business

- ❖ Non IT/IS methods to communicate with suppliers.
- ❖ Suppliers do not support E-Business implementation.
- ❖ Resistance to Change:
 - Structural Change.
 - Cultural Change.
 - Technical Change.
- ❖ Security issues.
- ❖ Payment tools.



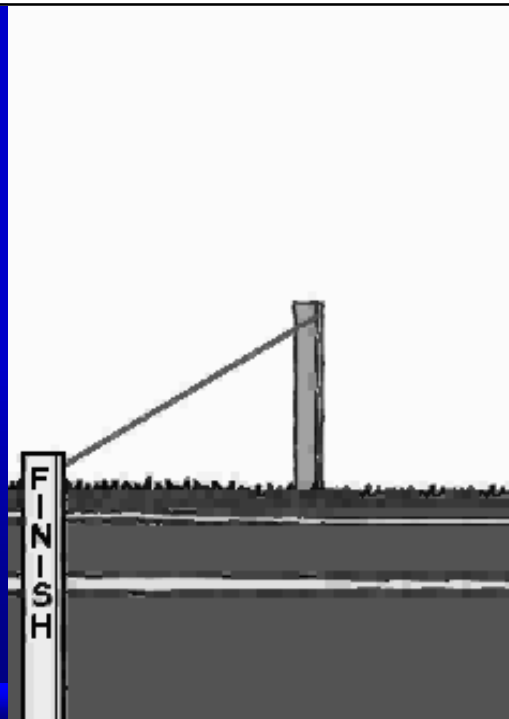
Results: Stages and types of Internet E-Business



Linkage to Enterprise Strategic Plans?

Conclusions

- * Benefits of Internet E-Business.
- * Examined Obstacles.
- * Investigated key driving forces.
- * High implementation costs.
- * Inadequate resources.
- * Insufficient external assistance.
- * Improving customer service.
- * Reducing costs.
- * Providing Business Intelligence.
- * Process simplification.
- * Taking faster decisions.



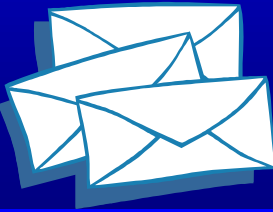
Recommendations



Recommendations

- ◆ Factors and degree of involvement during Stages (Planning, Development, and Implementation) and Types of Internet E-Business (Business-Business, Business-Customers and Business-Employees).
- ◆ Entrepreneurial factors affecting the introduction of Internet E- Business among businesses (size, type and management style).
- ◆ Change Management factors, such as Structural Change, Cultural Change and Technical Change.
- ◆ Importance of business culture and technology availability, during implementation of Internet E- Business.

For further information...



Tel: + 61 2 9514 3611
Fax: + 61 2 9514 3602
Mob:+ 61 417 285 687

E-Mail: Fawzy.Soliman@uts.edu.au

Dr. Fawzy Soliman
School of Management
University of Technology, Sydney
PO Box 123
Broadway, 2007
Australia



The



End

Pearson Education is a division of the international media group Pearson and was formed as a result of the recent merger of Addison Wesley Longman, Financial Times Management and Simon & Schuster's educational business. Pearson Education is the world's leading education business. From primary to postgraduate, teachers, students, professionals and parents can choose Pearson Education resources to meet their needs with confidence. Our imprints stand for quality, consistency and innovation in education throughout the world.

Pearson Professional Education is the largest supplier of Trade and Professional Computer books in the English language and consists of the well-known imprints: Adobe Press, Addison Wesley, Brady Games, Cisco Press, Hayden Books, JAMSA Press, Manning, Macmillan Technical Publishing, New Riders, Peachpit Press, Prentice Hall PTR, Que, and Sams. All of these books can be ordered in the at our offices in Amsterdam, or in the UK.

Pearson Education
Concertgebouwplein 25
1071 LM Amsterdam
The Netherlands

Tel: +31 (0) 20 575 5800
Fax: +31 (0) 20 664 5334

Pearson Education
Customer Services
PO Box 88
Edinburgh Gate
Harlow
Essex CM20 2JE
Tel: +44 (0) 1279 623 925
Fax: +44 (0) 1279 623 627

We are proud to be supporting two of our authors at the 3rd International Software Quality Week Europe '99, Suzanne Robertson, author of *Mastering the Requirements Process* and Martin Pol, author of *Test Process Improvement*.

Computer Associates International, Inc.

Developing & Deploying Applications in Internet Time.

Business has to respond to both these kinds of challenge, and respond quickly. Whilst business strategy may appear to be easy to change, the actual business processes embodied in our existing applications may not adapt readily to our new requirements. Building the flexibility to accommodate business change into applications is more than just a technology issue. It may also require a shift in an organization's development culture, in the development process, in the methods and best practices for development. Certainly there has never been a greater need for effectively managed development projects. Similarly, the need to deploy applications in "Internet time" is becoming increasingly critical.

Whether you are looking to design and build new applications or re-purpose existing ones, you will find much value in meeting with us.

Computer Associates International, Inc. (NYSE: CA), the world leader in mission-critical business computing, provides software, support and integration services in more than 100 countries around the world. CA has more than 17,500 employees and had revenue of \$5.3 billion in fiscal year 1999.

For more information about CA and CA Belgium, please call +32 2 773 28 11 or email info@cai.com.

CA's Web address is www.cai.com or www.cai.com/offices/belgium/belgium.htm



TESTFRAME

**Getting testing
under control**

Organizations operate in a turbulent market. More and more the need is felt to prevent interruptions of core business processes by computer failure. CMG has gained a wealth of experience in setting up test organizations using their full scale method for structured testing, named TestFrame. The emphasis in TestFrame is on the strategic support of the complete testing processes, from organization to test execution, in which all test products are put in place and related to one another. The underlying concept of TestFrame is that a test is set up right from the start with maintenance in mind, so that future checks can be carried out with the minimum adjustment to the test material. Because the testing products are re-usable, you will have recouped your investment after just a few re-tests.

For more information please contact the TestFrame Research Centre of CMG, telephone +31 348 45 40 00, fax +31 348 45 40 13, e-mail testframe@cmg.nl, Internet: www.testframe.com.

CMG
Information Technology

Organizations are stronger with CMG.

CMG FINANCE | CMG TRADE, TRANSPORT & INDUSTRY | CMG PUBLIC SECTOR | CMG TELECOMMUNICATIONS & UTILITIES | CMG BUSINESS SERVICES

TestFrame

Getting testing under control

CMG plc is a leading European IT services company, providing business information solutions through consultancy, systems and services to clients world wide. Established in 1964, CMG now operates in more than 40 countries from its bases in the UK, The Netherlands, Germany, France and Belgium. The company is listed on the London and Amsterdam stock exchanges. CMG supplies services and products in the finance, trade and industry, transport, telecommunications, energy and public sectors. The Group also provides managed information processing services, including networks, payroll and personnel.

Over the years, CMG has gained considerable experience in the setting-up of testing procedures and testing organisations. Based on this experience, CMG has developed an open test method **TestFrame**[™]. TestFrame is based on three principles: *Fitting*, *Structuring* and *Tooling*. First, the process must always be appropriate to the organisation in which it is to be adopted: it must fit. Second, testing can only offer quality control if it is approached structurally and allows for ready maintenance. And third, the use of tooling is essential to any modern testing process. This test method leads to re-usable test products so that future checks can be carried out with minimum adjustment to the test material.

CMG is dedicated to helping its clients and their people become more successful through the quality of its services and staff. Strong employee commitment ensures the Group's long term success and hence the success of its clients.

More information:

For more information please contact the

TestFrame Research Centre of CMG,

e-mail : testframe@cmg.nl

telephone : +31 348 454000

fax : +31 348 454013

LE COMMERCE
ÉLECTRONIQUE
OUVRE LES
PORTES DU MONDE
ENTIER À VOTRE
ENTREPRISE.
NOUS VOUS
PERMETTONS DE
SAISIR CETTE
OPPORTUNITÉ
AVEC LA
TECHNOLOGIE
QUE VOUS
POSSÉDEZ DÉJÀ.

Yes. We're
OPEN

*Découvrez comment nos logiciels et nos services peuvent aider votre entreprise à profiter
des avantages du commerce électronique en visitant notre site www.compuware.be*

COMPUWARE
What do you need most?sm

Compuware Corporation

Farmington Hills, MI

800-521-9353

www.compuware.com

Compuware Corporation offers software tools and professional services that help the world's largest companies more efficiently build, maintain and enhance their most critical business applications. Our unique combination of productivity tools and services has helped make us both the world leader in software testing and quality assurance and the world's fifth largest independent software vendor.

Compuware offers our 14,000 customers a one-vendor solution that extends across the application lifecycle, from development, through testing, and managing application service levels in production. Our tools and services extend across the enterprise, whether your application is mainframe, distributed, and/or web-based. Compuware's testing tools include QACenter, NuMega, File-AID, XPEDITER and EcoSYSTEMS.

***ErgoLight* Company and Product Description**

ErgoLight Usability Software

6 Giv'on St.,

Haifa 34335, Israel

Phone: +972-4-826-3012, +1-877-Use-Ergo (+1-877-873-3746)

Fax: +972-4-825-8199

Email: info@ergolight-sw.com

Web: www.ergolight-sw.com

Are you a developer of a productivity-critical system? A safety critical system? Or, a revision of a not-so-friendly system?

If the answer to any of these questions is yes, then you NEED ***ErgoLight*** tools. Why? Because common method and practices of usability testing target product learnability, which is not your project main concern.

ErgoLight Ltd. develops and markets software tools and services that enable Windows application developers to get the **user feedback** via the internet, to conduct **user testing** that focus on **user productivity and reliability** and to obtain **objective measures** of the **product usefulness**.

Are you concerned about the user productivity, operation reliability and long range product credibility? The single most important factor that determines these concerns is the software's capability to respond gracefully to user errors. ***ErgoLight*** tools are the only means available today that provides the information required to detect and evaluate critical user errors.

Looking for **objective measures** of product quality? ***ErgoLight's usage profiles*** show which GUI controls are used often and which are not used at all. In addition, the profiles enable you to pinpoint activities that consume most of the product operation time and to understand the reasons why.

Does your customer support understand the user feedback? Much of the support time is spent on user complaints about unexpected system behavior, which actually results from undetected user errors. By differentiating between GUI design flaws and software bugs, ***ErgoLight*** tools will save your programmers wasted debugging time and reduce your support costs.

ErgoLight Ltd., 6 Giv'on St.,
Haifa 34335, Israel
Email: info@ergolight-sw.com
Web: www.ergolight-sw.com

Tel: +972-4-826-3012
Fax: +972-4-825-8199
USA: 1-877-Use-Ergo
1-877-873-3746

e-Strategy

e-Strategy is a business consultancy and software VAR based in Brussels, Belgium, specializing in analytical tools for business and project management support. Interested participants are welcome to visit our booth in the Exhibit Hall to see an in-depth demonstration, pick up product literature, or receive a complimentary copy of the RiskDRIVER CD-ROM.

RiskDRIVER is a European web-based initiative to promote best practices in project risk management. The RiskDRIVER website, www.riskdriver.com, is a one-stop source for risk management practitioners, expert and novice.

RISKMAN is the only software package to provide thorough support of a risk management methodology and full integration of risk management in all phases of project management. RISKMAN provides a systematic approach to project risk management, guiding the user through the 5 steps of the RISKMAN methodology:

- Risk Identification: the Risk Catalogue facilitates the identification and classification of risks, together with their respective causes and mitigation actions. The Catalogue supports checklists, risk sheets, and transfer of risk data to and from projects.
- Risk Assessment: RISKMAN lets you choose an impact scale (cost, duration, quality) and determine unacceptable risks in terms of financial loss, delays, defects, . Use RISKMAN's budget management function to calculate and display graphically your budget for risk mitigation and provision for contingencies.
- Risk Action Plan: RISKMAN helps you define mitigation actions for unacceptable risks, which can be planned as tasks in Microsoft Project. Advanced Monte Carlo simulation takes into account uncertainties to analyse the effectiveness of mitigation actions and estimate the actual cost and end date of the project.
- Follow-up: RISKMAN makes it easy to track multiple risks, causes, actions, events and budgets throughout the project, producing customised reports in Microsoft Word and Excel formats.
- Knowledge capitalisation: the Risk Catalogue is an expanding knowledgebase that captures your corporate risk data and lets you share risk experience with project teams.

RISKMAN is a user-friendly Windows application that is fully compatible with Microsoft Project, available in English and French language versions. Download a 30-day evaluation copy from the RiskDRIVER website at www.riskdriver.com/riskmantool. Registered contributors to the RiskDRIVER site may also download a fully functional copy of the RISKMAN Catalogue.

EWO Software, Inc.

Company Overview - For over 30 years Information Systems organizations in the Fortune 2000 have had to deal with the programming staff having to sift through the massive applications programs and data to assemble the complete package to do either maintenance or write a major modification. EWO has solved this problem by providing a complete and comprehensive software package called "Prescient®", meaning "forethought". The Prescient family of products consists of eight modules individually designed to solve the various problem of manually searching for all the constituent elements of an application or production job, inventory analysis and program cross references to mention a few.

The company markets the "Prescient" product Worldwide to the MVS user base. The company believes that with the modular orientation of its products directed towards particular Information Systems department solutions, and further, provided at reasonable cost, with superior service/support availability is a winning approach and will continue to be successful.

The concept of the Prescient Suite is to essentially automate the analysis and setup function that is currently being done manually by Mainframe applications programmers in every data center in the world having the IBM MVS or the new OS/390 Operating System. The software solutions from the Company need to be presented and sold using a consultative technical approach. The MVS market needs to be trained in the use of the tools and shown exactly how the tools work. The good news is that this takes only a two-day period to get the applications staff fully productive with the software.

The Company's experience has been that the customer looks at the Prescient products as beneficial because the products make their business more productive and therefore more profitable. In addition, it has been readily accepted because it truly helps the Information Systems group moves faster through its applications as they solve the Year 2000 problem and/or provide maintenance for their applications.

EWO has established a reputation in the marketplace for developing high value products that deliver cost effective, timesaving products which are sold at fair market value for uses in the Mainframe and Internet market. We have achieved this by developing cutting edge product development with a close understanding of market trends and needs.

Business Description - EWO was founded in 1991. It was organized to provide innovative software products for the IBM - MVS Operating System and those companies that use this system. This is predominately the Fortune 2000 worldwide. EWO started out meaning "Easy Way Out" which referred to the ease of use and the problem solving nature of the initial product called Documentor. Because of the broad base of the company products in the MVS environment, the EWO acronym can best be referred to as ***Enterprise Wide Operation***.

The PreScient Product Line - The major modules included in the PreScient product line are identified as follows by their application, which provide greater control in the SCM market. The product lines all share in the core technology which consists of the "smart parser". There are also other modules within PreScient that will be addressed later in the plan.

1. Documentor

Documentor is a complete analysis package for identifying the overall program inventory in the customer's MVS – OS/390 system. Proper inventory identification is a major problem today that surfaced as a result of the Year 2000 problem. It will address the inventory requirements of the Fortune 2000 and will allow companies to truly exercise a "clean management" system.

2. Replicator

Replicator is an extension of the Downloader, however it includes the ability to handle both programs and data. It is useful in setting up test beds for applications and assures that all the elements are identified for the subsequent test. Using Replicator can save hours and sometimes days of programmer time in setting up an accurate test plan. EWO Software is the first to market a product like Replicator.

3. LoadSync

LoadSync represents a new approach to the not so obvious problem of out-of-sync load libraries. LoadSync validates and verifies the actual synchronization of library load modules in the effort to identify potential date mismatches of static and dynamic call programs which are a result of the serial program compiling process. This is perceived to be a major problem particularly in view of the Year 2000 problems facing everyone.

4. Downloader

Downloader has been a valuable tool for customers having to move applications from point A to point B. This product allows applications to be moved in complete form (JCL, programs, copybooks, etc) from Mainframe to Mainframe or from Mainframe to alternate platforms for such use as code remediation or re-hosting. It addresses the needs of the entire range of OS/390 customers as do all the modules in the PreScient software suite.

Services, Custom Development/Training/Consulting - EWO Software also provides consulting services for Mainframe support.

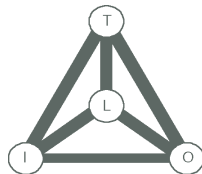
Technology - Proprietary Technology - The software products are proprietary in their design and in their software architecture and methodology. The software is protected by copyrights and a trademark has been applied for. In addition, the software is shipped as executables and the source code is present only on our development system. Software routines are used to provide time dependent usage and can be activated by use of unique serial numbers and passwords.

quality in structured testing



Gitek is the exclusive distributor of TMap® and TPI® in Belgium.

TMap® (Test Management approach) is a structured approach for testing software applications. The approach is based on four cornerstones:



- Life cycle
- Techniques
- Infrastructure and tools
- Organisation

TPI® (Test Process Improvement) is the ongoing process to improve the test processes. The TPI®-model is a practical step-by-step guide to upgrade the quality of your tests.

Our services:

- Test advice
- Structuring test processes
- Improving test processes
- Training and coaching in testing
- Test planning and test management
- Test design and test execution



Gitek, Sint-Pietersvliet 3, 2000 Antwerpen, tel.03/231.12.90, fax 03/226.10.83, e-mail gitek@gitek.be, visit www.gitek.be

Gitek nv

Gitek nv was founded in 1986 . Over 100 employees support customers in delivering quality information systems. Gitek has also specialised in testing software applications using TMap® the structured testing approach developed by its partner IQUIP. With 40 professional testers it is the leading testing company in Belgium. Gitek has the exclusive rights for Belgium for TMap® and its related products : TPI® (Test Process Improvement), TAKT♥ (Test Automation) and TSite® (Test Laboratory).

Gitek nv

St.-Pietersvliet 3

B-2000 Antwerpen

Belgium

Phone : 00/32/3 231 12 90

Fax : 00/32/3 226 10 83

e-mail : gitek@gitek.be

www : www.gitek.be

INTEGRI

Leuvensesteenweg 325
B-1932 Zaventem, Belgium
Tel 32.2.712.07.50, Fax 32.2.712.07.67

info@integri.be

www.integri.be

PRODUCTS AND SERVICES

Integri provides its clients with two different services : On the one hand, Integri can organise and execute acceptance and certification tests for its clients, acting as an independent test company. The client provides Integri with detailed functional specifications of a project. From these functional specifications, Integri derives the tests, executes them on the unit under test and issues a test report. On the other hand, Integri can provide the client test team with a full test environment. A number of off-the-shelf tools are available, such as THALES (for testing and emulating smartcards) and CERTO (for testing and emulating host interfaces). However Integri's software can easily be adapted to the client's specific requirements. Integri's tools Typical systems and projects for which Integri's test technology and methodology is suited for are : **smartcard masks, smartcard terminals, security systems, authorisation host interfaces**, etc...

CLIENTS

Integri's customers are large national and international payment system providers, smartcard and payment terminal manufacturers, based in various European countries. Typical clients are Banksys Belgium, APSS Austria, Europay International, EPCI Belgium, Giesecke & Devrient Germany, Graphium Denmark, Incard Italy, Telekurs Switzerland, SSB Italy, SWIFT, TSP Italy, etc.

Integri also started providing services to **GSM operators**, as the **THALES test tool** have been extended for the **GSM 11.11, GSM 11.14 and GSM 11.17** standards.

IPL Information Processing Ltd

Software quality can only be assured through proper application of a number of activities and techniques. One of the foremost of these is of course testing, and in particular testing at the early stages of the code production process. These activities go by the general name of 'unit' (or 'module'), and 'integration' testing. The problem tends to be that these activities are time-consuming. The way out of this dilemma is to use tools which can to a reasonable degree automate the activities, and that is the purpose of the IPL tools, AdaTEST (for Ada) and Cantata/++ (for C/C++).

These products have achieved outstanding reputations for themselves in their particular field since their initial release eight years ago. In the various fields of civil and military avionics, air traffic control, space, telecommunications, automotive and railway, and all areas where the need for reliability dominates AdaTEST and Cantata have played their roles.

Specifically the products allow software components to be fully executed in an automated and repeatable fashion, and providing full isolation (where appropriate) from external software. These tests can be executed in both host (PC and Unix) and target environments. The tools also allow coverage of these tests to be measured, thus allowing the developers to ascertain a 'confidence' factor in their testing. Finally, a wide range of static analysis metrics facilities allows developers to monitor their source code against various criteria to ascertain its 'quality' and maintainability.

At QWE'99 IPL will be exhibiting all their tools and services. The Vendor Technical Presentation will be devoted to a short presentation of Cantata++ for testing C++.

IPL Information Processing Ltd
Eveleigh House
Grove Street
Bath BA1 5LR
United Kingdom

Tel (DDI): +44 (0)1225-475114
Fax: +44 (0)1225-444400
E-mail: iang@iplbath.com
World Wide Web <http://www.iplbath.com/tools>



**Should management
be dependent
on information systems?**



In this day and age, organizations are often hampered by software bugs when using information systems. One of the reasons for this is that the integration of business processes and related innovations, such as the Internet, electronic commerce or knowledge management, produce new challenges time and again. This could, for instance, result in the requirement that all components of tailor-made software should be able to function together. And this would make prior testing of systems not only essential but would also demand a new approach to testing.

www.iquip.nl



At IQUIP we are used to thinking things through. For this reason we are always searching for new testing methods. With TMap®, IQUIP has developed a successful, structured test approach. TMap® is already being used by more than 200 companies in the Netherlands, Belgium and other European countries. In addition, IQUIP offers TAKT (automated testing), TSite® (Test Factory) and TPI® (stepwise improvement of test processes) which provide excellent opportunities for securing the future of testing organizations. IQUIP is constantly seeking integrated, total solutions for its customers, under which each organization remains flexible under all conditions. And where management can have complete faith in the information system, in order to take advantage of market developments as they occur.

Would you like to know more about what our vision of the future could mean for your organization? Then please call Martin Pol, + 31 (0)20 660 66 00 for further information.



IQUIP Informatica B.V.

Since 1972 IQUIP Informatica B.V. has been supporting organisations in carrying out their core processes. With its 1400 employees IQUIP does this by constructing, maintaining, testing and implementing application software systems. IQUIP increasingly concentrates on carrying out assignments with result responsibility. In testing, IQUIP achieves this through her dedicated division Components & Testing (300 employees), using the structured testing approach TMap®. TMap® was developed by IQUIP itself and has become a widely used international standard. TMap® related methods such as TAKT© (test automation), TSite® (test laboratory) and TPI® (Test Process Improvement®) have recently completed the product range. A dedicated R&D team is continuously improving these methods and, if required, develops new products.

IQUIP Informatica B.V.

Head Office: Wildenborch 4
Address: Postbus 263
1110 AG DIEMEN
The Netherlands
Phone: +31 (0)20 660 6600
Fax: +31 (0)20 698 1437
E-mail: info@iquip.nl
WWW: www.iquip.nl

McCabe & Associates (UK)

Chancery Court
Lincoln Road
High Wycombe
Bucks
HP12 3RE
Tel: +44 (0) 1494 463 233
Fax: +44 (0) 1494 463 288
Email: sales@mccabe.co.uk
<http://www.mccabe.com>

McCabe & Associates is an international leader in software solutions for improving the quality and reliability of enterprise software applications. Based on over twenty years of research and experience in software quality, testing and re-engineering, McCabe IQ (TM) is an integrated approach to building quality into your software development lifecycle. McCabe IQ combines a strong theoretical foundation with practical, visual tools to help organisations:

- Accurately assess software quality, complexity, and testing requirements
- Pinpoint potential problems and high-risk areas
- Eliminate redundant code
- Thoroughly test applications
- Validate coverage of high-risk areas

McCabe IQ (TM) supports developments in Java, C++, C, VB, COBOL, FORTRAN and Ada

Since 1977, we have been working closely with our customers to improve the quality of large-scale, mission-critical software. Many of our world most influential corporations and government organisations have used our products successfully to test, re-engineer, and verify the quality of over 30 billion lines of mission-critical code. Today, we are one of the most highly respected vendors of products for source code analysis and testing. Our structured testing methodology has been adopted and published by the National Institute of Standards and Technology(NIST).

Solidly based on source code analysis technology, McCabe IQ integrates the build, test and change phases of software development. By linking these processes through advanced visualisation, industry standard metrics and dynamic monitoring, McCabe IQ raises the quality standards of your deliverables, reduces your overall development costs, decreases your time to market and lets you focus your resources where they will have the greatest impact.



*Does your application expand your future
or confine it?*

**TEST FOR
SCALABILITY.**

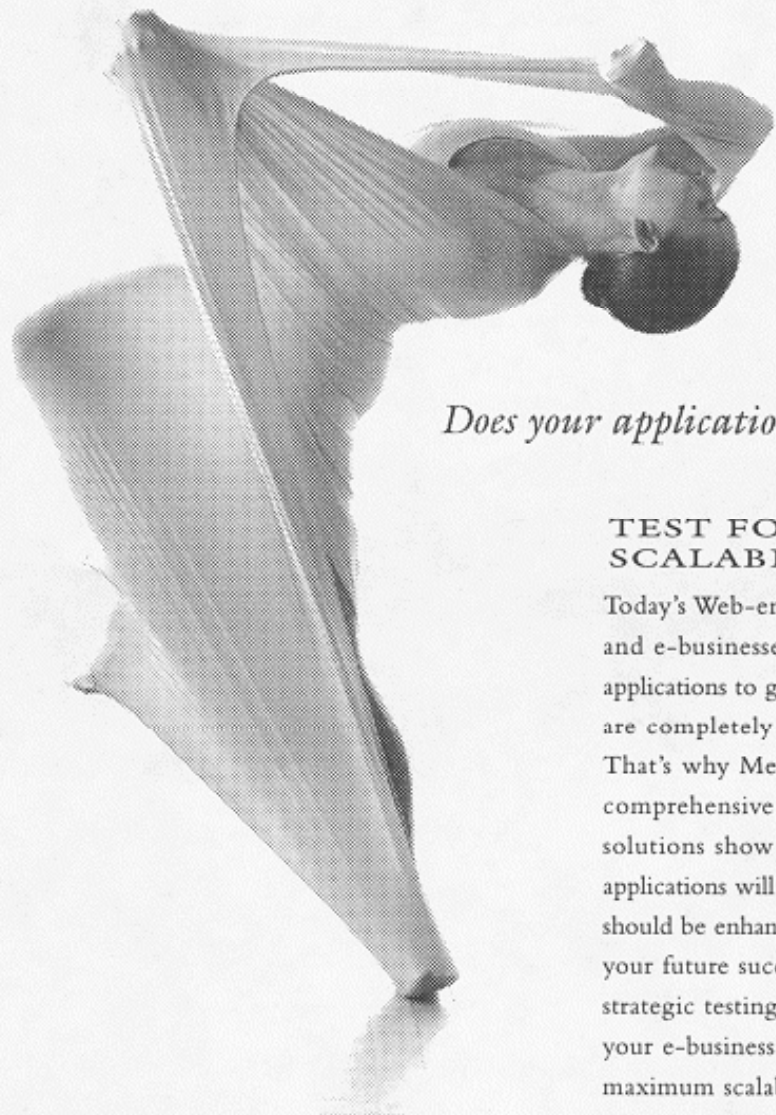
Today's Web-enabled enterprises and e-businesses can subject your applications to growth demands that are completely unprecedented. That's why Mercury Interactive's comprehensive application testing solutions show precisely how your applications will scale, and where they should be enhanced to accommodate your future success. Learn how a strategic testing initiative can help your e-business applications achieve maximum scalability, extraordinary user satisfaction, increased functionality, and rapid deployment.

Visit us at www.merc-int.com



MERCURY INTERACTIVE

The test of a great business.



*Does your application expand your future
or confine it?*

**TEST FOR
SCALABILITY.**

Today's Web-enabled enterprises and e-businesses can subject your applications to growth demands that are completely unprecedented. That's why Mercury Interactive's comprehensive application testing solutions show precisely how your applications will scale, and where they should be enhanced to accommodate your future success. Learn how a strategic testing initiative can help your e-business applications achieve maximum scalability, extraordinary user satisfaction, increased functionality, and rapid deployment.

Visit us at www.merc-int.com



MERCURY INTERACTIVE

The test of a great business.

Mercury Interactive

1325 Borregas Avenue
Sunnyvale, CA 94089 USA

T: (800) TEST911

(408) 822-5200

F: (408) 822-5300

For local offices worldwide, visit our Web site at www.merc-int.com

Mercury Interactive Corporation develops, markets and supports a suite of solutions that automate testing, quality assurance and application performance management, from Internet/e-business applications and front office applications—like sales force automation or help desk—to back office ERP applications.

We help people deploy software with confidence.

And we offer the solutions, the support, the technology and the commitment to help you.

TestDirector®

An integrated enterprise application for organizing and managing the entire testing process.

By combining planning, execution and defect tracking with open test architecture and a central repository, TestDirector consolidates and manages the testing process to determine application readiness.

WinRunner® and XRunner®

Enterprise functional testing tools that verify Windows and UNIX applications work as expected. By capturing and replaying user interactions automatically, WinRunner and XRunner identify defects and ensure that applications work flawlessly the first time and remain reliable.

LoadRunner®

A load testing tool that predicts system behavior and performance. It exercises the entire enterprise infrastructure by emulating thousands of users to isolate problems, optimize performance and accelerate application deployment.

QuickTest™

A family of business process testing tools designed to be employed by the end-user.

TestSuite®

A set of product suites designed for specific market segments, including:

- TestSuite Enterprise—for managing testing procedures for ERP applications
- TestSuite 2000—for managing Year 2000 testing challenges
- TestSuite Euro—for managing Euro currency conversion issues

Astra™

A family of e-business testing tools, including:

- Astra SiteTest, a load-testing tool
- Astra SiteManager, a comprehensive visual Web site management tool suite
- Astra QuickTest, an icon-based functional testing product

Mission statement of ps_testware:

“To offer the best solution to quality problems of computer systems by using its test expert knowledge in a professional way.”

- *Best solution: the solution that provides the highest contribution.*
- *Computer systems: hardware as well as software under all forms, such as IT, automation, embedded systems, etc.*



- *Test expert knowledge: the intellectual asset of ps_testware, a profound and complete knowledge regarding verification and validation (testing).*
- *Professional: the courage to really provide what has been promised.*



ps_testware

The mission of ps_testware:

To offer the best solution to quality problems of computer systems by using its test expert knowledge in a professional way.

- ✓ Best solution: the solution that provides the highest contribution.
- ✓ Computer systems: hardware as well as software under all forms, such as IT, automation, embedded systems, etc.
- ✓ Test expert knowledge: the intellectual asset of ps_testware, a profound and complete knowledge regarding verification and validation (testing).
- ✓ Professional: the courage to really provide what has been promised.

Don't hesitate and visit our website: <http://www.pstestware.com>

Q-Labs

Q-Labs is dedicated to providing software engineering solutions to the industry, enhancing software development capabilities. Q-Labs' expertise is as a change facilitator, which includes providing 'state-of-the-art solutions' as well as supporting dissemination of 'industry standard best practices.'

Our approach is to keep our customers steps ahead in software engineering.

Q-Labs products and services include:

- o Software Management
(Business Analysis, Software Process Improvement, ROI Calculations, ...)
- o Software Process Improvement
(SW-CMM-based Software Process Improvement, SE-CMM, Measurements, ...)
- o Technology Transfer and Deployment services (Inspections, UML, SDL, ...)
- o Cleanroom Software Engineering
(Statistical Usage Testing, Sequence Based Specifications, Incremental Development)
- o Humanics (Teamwork, P-CMM)
- o Software Acquisition services (SA-CMM, Supplier Evaluations, External QA, ...)
- o Test Support (Daily Builds, Automated test)
- o Contract Development Services
(software specification, development and test for customer projects)

Our goal is customer success. For example, Q-Labs in its role as the primary provider of SPI support for the Ericsson System Software Initiative, helped Ericsson save 60 million USD in 1997 due to improved software quality.

Q-Labs works with a number of customers around the world, including Alcatel, Bosch, CTI, Ericsson, IBM, Siemens, and the U.S. Army.

Q-Labs was founded in 1989 and has over 60 employees at offices located in Sweden, Germany, US, Ireland, and Norway. We have a very strong international network and a strong capability to support global customers. Q-Labs is a joint venture between DNV and Ericsson.

For more details, visit us at www.q-labs.com or send write us at info@q-labs.com.

Qualityhouse

Qualityhouse is the Netherlands' leading independent company aimed at Quality-assurance in ICT. Qualityhouse does not involve itself in software development whatsoever and is thus able to use its unique position in the market to offer a truly independent quality assessment. The range of services offered by Qualityhouse include:

Testing software

Qualityhouse supports software testing by placing testers as consultants within projects. These consultants are highly educated and skilled specialists in the field of software testing. Projects can be performed in-house at the client's site. Additionally Qualityhouse offers the possibility of outsourcing the entire test process.

Improving test processes

As a specialised software testing company, Qualityhouse can help you improving your test processes. Qualityhouse can provide assistance in adjusting current test processes to a changing environment or in designing the structure for an entirely new test process.

Improving software development processes

Qualityhouse has extensive experience in assessing software development processes. Using the Capability Maturity Model, assistance can be given in improving software development cycles. Qualityhouse can effectively lead diverse groups, working towards successfully establishing a more efficient development process.

Training

Qualityhouse shares its in-depth knowledge and experience in software testing by offering a variety of courses and workshops. A full six week in depth course on software testing can be followed by prospective software testers. Additionally in-house courses are given on a selection of relevant software testing subjects.

QUALITYHOUSE

Qualityhouse is the Netherlands' leading independent company aimed at Quality-assurance in ITC.

Qualityhouse focuses on:

- testing software
- improving testprocesses
- improving software development processes
- courses on testing

Qualityhouse supports software testing by placing testers as consultants within projects and by offering the possibility of outsourcing the entire testprocess.

Qualityhouse does not involve itself in software development whatsoever.

As a result of this unique position

Qualityhouse is capable of offering a truly independent quality assessment.

Qualityhouse BV,
P.O. Box 5138, 1410 AC Naarden, The Netherlands
Tel: +31 (0)35 699 02 22 Fax: +31 (0)35 699 02 29
E-mail: info@qualityhouse.nl

Rational Software Corporation

Rational Software (NASDAQ: RATL), creator of the Unified Modeling Language (UML), is the leading provider of a solution that unifies proven software development principles, tools, and services to improve the productivity of project teams and individuals. Rational's products span the critical activities of requirements management, visual modeling, testing, and configuration and change management.

Rational's mission is to ensure the success of customers who depend on their ability to develop the software upon which their businesses depend. Rational enables its customers to achieve business objectives by turning software into a source of competitive advantage, decreasing time-to-market, reducing the risk of failure, and improving software quality.

Rational's comprehensive solution unifies proven principles of software development, an integrated family of market-leading tools, and technical consulting services into a set of software Best Practices applicable through the development lifecycle by all members of a development team. Rational's products can be purchased and used individually or integrated with other Rational products, leveraging the power of each individual product.

Rational has more than 1900 employees worldwide, with corporate headquarters in Cupertino, California. Major development centers are located in California, Massachusetts, Oregon, Colorado, North Carolina, Washington, Pennsylvania, Sweden, and India. For more information, visit Rational's Website at www.rational.com <<http://www.rational.com>> .



Before taking another step, make sure your mission-critical applications are bug-free

Reasoning, Inc. is the leading provider of automated software inspection services that finds bugs in software fast and at 1/10th the cost of testing. Whether you are concerned about the reliability of mission-critical applications or eBusiness applications that must run flawlessly—24 by 7 by 365—Reasoning's highly automated Inspector Plus services can reduce the risk of application failures and enhance the quality of your software. Let Reasoning be your software reliability safety net.

To find out more about Reasoning and how to contact a Reasoning representative in your area, visit our web site at www.reasoning.com.



Reasoning Inspector Plus

Automated Software Quality and Reliability Services



System downtime resulting from undetected software defects cost organizations \$85 billion in 1998 despite the billions of dollars spent on software testing and maintenance.

To help enterprises and commercial software developers avoid the problems caused by software defects, Reasoning, Inc., the world leader in software inspection services, offers Inspector Plus.SM Inspector Plus augments resource-constrained software development and quality assurance teams by automatically pinpointing a wide range of software defects that can cause application failures and data corruption. Based on the results of the inspection, Reasoning's consultants also provide an assessment of the overall quality and reliability of an organization's software.

The service employs a highly automated software inspection toolset that is based on Reasoning's advanced Code-base Management System (CBMS). The toolset uses sophisticated software re-engineering technology and artificial intelligence to analyze, learn, and adapt to different coding styles. This unique capability combined with a proven software inspection methodology, enables fast and highly accurate inspections.

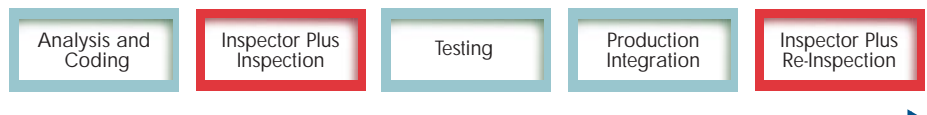
Inspector Plus:

- Identifies defects early
- Finds defects that testing misses
- Finds defects faster at one-tenth the cost of testing
- Verifies and validates software modifications
- Assesses compliance with corporate coding standards
- Improves reliability

• The Value of Inspector Plus Services

Inspector Plus services significantly enhance application reliability and data integrity by finding defects early in the development cycle. This approach reduces disruptions, coding rework, and recurring testing that prolong the development cycle and increase the chances of introducing new errors. Inspector Plus is extremely effective when used either before or after testing. The optimal approach is to inspect code prior to testing and to re-inspect during maintenance.

Where Inspector Plus fits in the development cycle

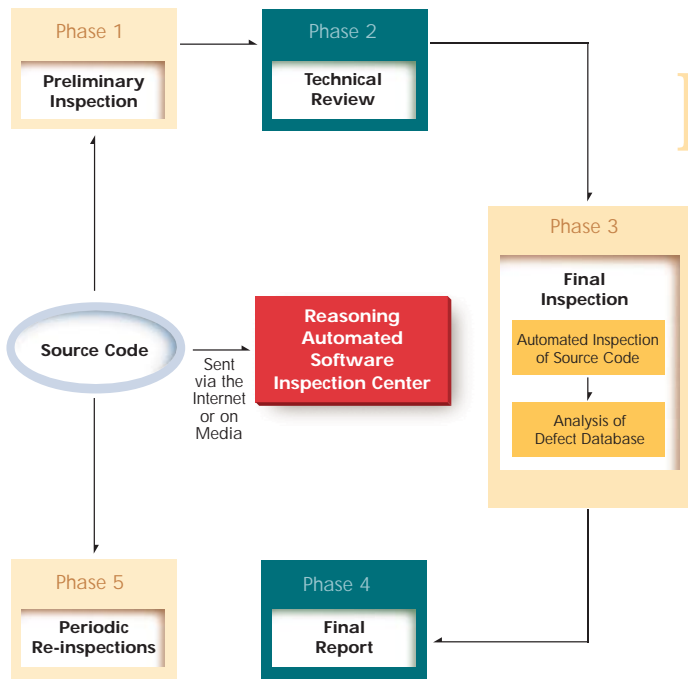


automated software inspection

Inspector Plus at Work

Inspector Plus combines an exhaustive, high-speed analysis of software with the expertise of analysts in a five-phase methodology that provides fact-based technical information and a results-oriented Action Plan. The automated software inspection process detects a broad range of software defects and identifies error-prone programs. In addition, Inspector Plus classifies the defects recorded in an organization's defect tracking system to determine their root causes and highlight areas that may require process improvement. Using the results of Inspector Plus analysis, Reasoning consultants prepare an Action Plan to improve the reliability of the software and decrease the time and cost of maintenance.

Inspector Plus five-phase methodology



Phase 1: Preliminary Inspection

phase 1

Certified inspection analysts using Reasoning's automated Inspector Plus technology start the inspection process by analyzing a sampling of source code to identify defects that could cause application failure, data corruption, incorrect behavior, or unpredictable results. Reasoning analysts check the preliminary computer-generated results and review potentially serious defects with the client.

Reasoning's automated inspections detect over 35 different types of defects, which can cause data corruption and system abends, including:

- Computational defects, such as size truncation or arithmetic overflow
- Data handling defects, such as uninitialized data elements or lack of end-of-file checks
- Interface defects, such as missing subroutine arguments or lack of return code checks
- Fragility defects, such as deeply nested IF statements or overlapping unstructured code
- Resource defects, such as memory leaks or improper memory deallocation

phase 2

Phase 2: Technical Review

After the preliminary inspection, Reasoning consultants collaborate with the client's IT management and application subject matter experts to identify and prioritize the focus areas of the final inspection. Focus areas may include specific types of defects, compliance to corporate coding standards, and process improvements. Objectives are set for broadening the scope of the final inspection to include an analysis of the defect database. Information gleaned from the defect database augments the inspection process and can have a significant impact on Reasoning's findings and recommendations.

Reasoning can customize Inspector Plus technology to detect violations of corporate coding standards including:

- Naming standards, such as defined field lengths and restrictions on first characters in names
- Formatting rules, such as enclosing literals in quotes and indenting certain clauses
- Size restrictions, such as the number of statements in a module
- Prohibited verbs, such as GO TO and STOP
- Documentation rules, such as requirements for documentation of program function and output messages

Phase 3: Final Inspection

The purpose of the final inspection is to generate an Action Plan to cover the following areas:

- Software Defect Action Items
- Software Fragility Action Items
- Software Process Action Items

Software Defect Action Items identify critical software defects that should be removed to improve the reliability of the application. These defects are identified by integrating the results of the code inspection and defect database analysis together with the feedback from the application development team.

Software Fragility Action Items identify those modules of an application that are most fragile by measuring each module's software fragility index (SFI). This index determines the likelihood that code will fail by analyzing code structure. The steps required to reduce its fragility is also determined.

Software Process Action Items include a set of recommended best practices that could reduce frequently occurring defects that are recorded in an organization's defect tracking database. Process recommendations may eliminate defects introduced at any point in the life cycle including requirements analysis, design, coding, and testing.

Phase 4: Final Report

In the final management briefing and report, Reasoning presents and documents the Action Plan. The plan describes Reasoning's recommendations for both software and process improvements.

phase 4

Comprehensive Reporting

- **Software Defect Summary Report:** identifies defects in each program that may cause it to function unreliably
- **Software Defect Cross-Tabulation Report:** describes the number and specific type of defects by program
- **Software Defect Detailed Report:** notes the file, line, defect class, statement, and data elements for each defect
- **Software Fragility Report:** identifies modules that are most likely to break as a result of maintenance actions
- **Defect Classification Report:** classifies the types of defects reported in the defect tracking database
- **Standards Compliance Report:** lists corporate standards violations by program and type of violation

Software Recommendations

The action plan defines specific defects in the code that could affect future output. Fragile modules that are at highest risk are also pinpointed. Reasoning delivers software recommendations that enable managers to focus their resources, improve the reliability of their software, and reduce its fragility.

Process Recommendations

Based on the results of defect classification and coding standards analyses, Reasoning recommends best practices that can reduce the occurrence of defects and improve compliance coding standards. Inspector Plus deliverables include a complete set of summary and detailed reports that provide the foundation for the recommendations documented in the Final Report.

phase 3



**Automated
Software Inspection**

Reasoning Inspector Plus



phase 5

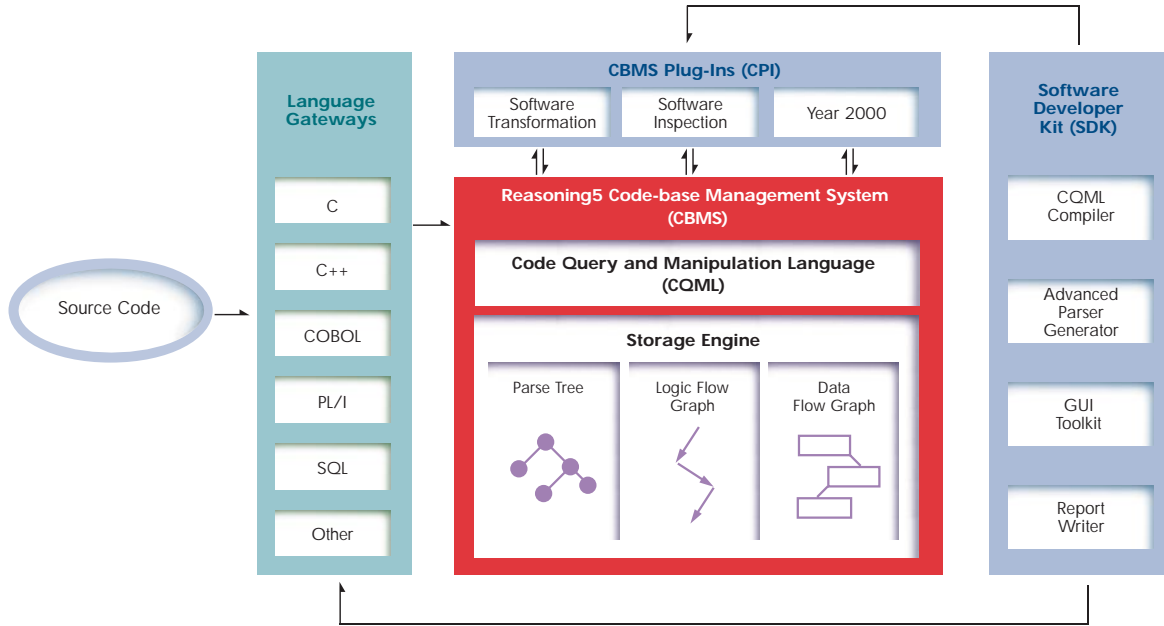
Phase 5: Periodic Re-inspection

Periodic Re-inspections leverage the clients' investment and the knowledge gained in the initial inspection process to ensure applications remain defect-free during redevelopment and maintenance.

Advanced Technology for Automated Code Inspections

Inspector Plus is one of many tool-based processes that are integrated with the Reasoning5™ CBMS—a revolutionary code object store and software analysis/transformation engine. Using the Software Developer Kit, Reasoning and technology partners have built many Language Gateways and CBMS Plug-Ins for automated software inspection, Y2K compliance, and the transformation of applications to new platforms.

Reasoning Code-base Management System



Language Support

- C
- C++
- COBOL
- PL/I

Summary

Reasoning offers Inspector Plus services to help software development organizations produce highly reliable software faster and at a lower cost—without placing new demands on in-house IT departments. These services complement testing and are more accurate and scalable than manual software inspections. Inspector Plus provides a safety net that helps protect enterprises and commercial software developers from the serious and costly consequences of software failures.



Reasoning, Inc.
700 East El Camino Real
Mountain View, CA 94040
U.S.A.

Tel: +1 650.429.0350 | Fax: +1 650.429.0222 | Web: www.reasoning.com | Email: info@reasoning.com



Reasoning/Inspector Plus

Automated Software Quality and Reliability Services

Despite the billions of dollars spent on software testing and maintenance, defects in software frequently cause serious and costly business disruptions.

To help enterprises and commercial software developers avoid the problems caused by software defects, Reasoning Incorporated, the world leader in software inspection services offers Inspector Plus™. Inspector Plus assesses the overall quality and reliability of software and pinpoints a wide range of software defects in applications that can cause failures and data corruption.

Inspector Plus combines the power of computers with the expertise of analysts in a five-phase consultative process designed to provide fact-based technical information and a results oriented action plan. It relies on an automated software inspection process to detect many classes of software defects and to identify error-prone programs. In addition, Inspector Plus classifies defects recorded in an organization's defect tracking system to determine the root causes of defects and highlight areas that may require process improvement. Using the results of its analysis, Reasoning's consultants prepare an Action Plan to enhance the reliability of the software and decrease the time and cost of maintenance.

Inspector Plus can help your software development and quality assurance teams develop and maintain highly reliable software faster and at a lower cost. Implemented as a service, Inspector Plus is faster and more cost-effective than in-house code inspections because it uniquely combines advanced technology, software inspection experts, and a repeatable software inspection process. It helps enterprises reduce the risk of costly software failures, before testing, and long before they impact the enterprises where they are deployed.

or service names may be trademarks of the companies with which they are associated.

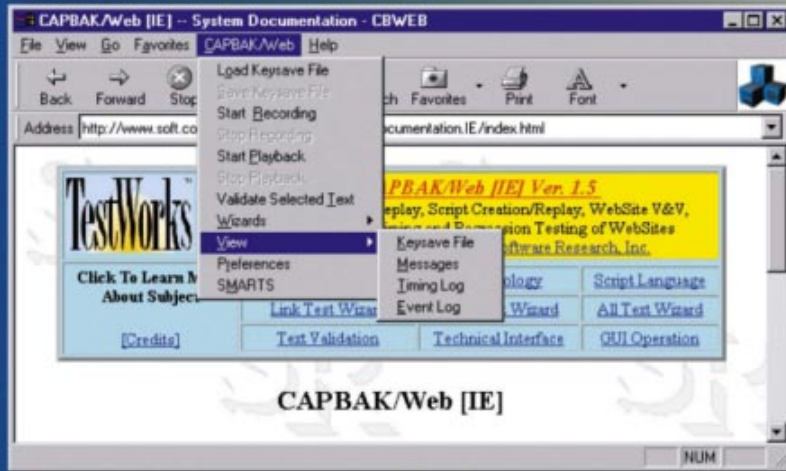
Reasoning

"Reasoning provides a new class of e-services to enhance the quality and reliability of Internet and legacy applications.

Reasoning's Inspector Plus uses a combination of automated software inspection technology and a repeatable methodology to rapidly inspect software for defects that cause software failures and data corruption.

Unlike conventional solutions, Reasoning's Inspector Plus services does not require large investments in staff, tools, and training. The benefits are more reliable software, reduced testing costs and faster time to deployment."

Make your website 100% reliable and your e-business 100% successful! — automatically



You can't afford
a broken website and
frustrated customers.
Find out before they do.

No miracles. Just solid solutions.

Quality Testing of Websites:

- Recorded & Scripted Playback
- 100% Object Mode + TrueTime
- Fully Synchronized Playback
- Content Validation
- Helpful Test Wizards
- SMARTS™ Interface
- Site Verification



Software Research, Inc. We know how.
901 Minnesota Street, San Francisco, CA 94107
www.soft.com (800) 942-SOFT info@soft.com



Software Research, Inc.

901 Minnesota Street
San Francisco, CA 94107 USA
<http://www.soft.com>

Software Research, Inc. is a leading edge company that develops, markets and supports the **TestWorks** suite of automated software and website testing tools. We work in partnership with top level managers, developers and QA organizations to assist them in meeting their product goals. We deliver industry-leading products, educational seminars, product training and technical support.

TestWorks is a fully integrated suite of tools that help ensure the quality of software products. **TestWorks** offers an end-to-end solution that covers all aspects of the process life cycle, including: test design and development, test data generation, test execution and evaluation, reporting and test management, code comprehension, coverage analysis, metrics and maintenance, and the development and maintenance of WWW technology.

Our latest tool, **CAPBAK/Web [IE] Ver. 1.5**, is a **TEST-ENABLED WEB BROWSER** for Windows 95/98/NT, aimed at validation and verification features and activities of complex websites. **CAPBAK/Web** is an object-oriented capture replay system, based on the Microsoft Internet Explorer [IE] browser. Webmasters now have a highly reliable test engine to help determine that a website is acceptable and credit card transactions are functioning properly. In the age of e-commerce, with multi-million dollar web transactions hanging in the balance, **CAPBAK/Web** will help assure the reliability and quality of e-commerce operations.

At **Software Research**, we believe that high quality products and services empower our customers to be successful. With our broad suite of products, our customers continually enhance their QA process, reduce costs and schedule overruns, and improve their overall product quality.




Testing E-Business Applications



Regis Mauger
European Field Market Manager

www.merc-int.com



Topics


- **Introduction**
- **How to capture Business Processes**
- **How to use Business Processes captures**
 - To do Functional Testing
 - To do Load Testing
 - To monitor the Service level delivered in production
- **Mercury Interactive Background**



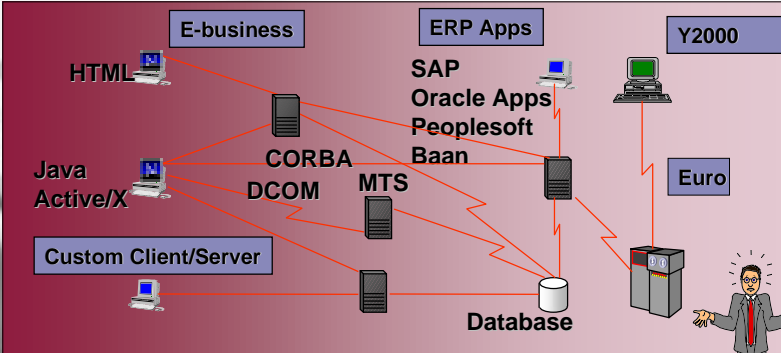
Introduction



www.merc-it.com



The Reality: Enterprise Complexity



- Heterogeneous Systems
- Variety of tools used in implementations
- Shortage of IT resources

Mercury Interactive's Testing Solutions

Open Test Architecture : Integrate Other Application Lifecycle Mgmt Tools

| | | |
|--|---|---|
| <p>Web/e-business</p> <p>Astra for: Fast, simple web testing</p> <p>WinRunner & LoadRunner for: Testing Enterprise Web-based Applications</p> | <p>Packaged Applications</p> <p>WinRunner & LoadRunner for: Baan SAP Oracle Applications Peoplesoft</p> | <p>Y2K/Euro</p> <p>WinRunner & LoadRunner for: Y2K / Euro Testing</p> |
| <p>Custom Client/Server</p> <p>Regression & Functional Testing: WinRunner Load and Performance Testing: LoadRunner</p> | | |
| <p>Test Management: TestDirector</p> | | |

Astra & TestSuite Enterprise 6.0

- Astra - Fast, simple web testing**

 - Astra QuickTest, Astra LoadTest, Astra SiteManager
 - ActiveScreen technology simplifies testing
 - Architected from the ground up for web testing
 - First testing tools you can "try & buy" on the Web
- TestSuite Enterprise 6.0 - an Integrated suite of Enterprise testing tools...**

 - TestDirector 6.0 for test management
 - WinRunner 6.0 for functional & regression testing
 - LoadRunner 6.0 for load testing


...ensure higher reliability and a positive end-user experience



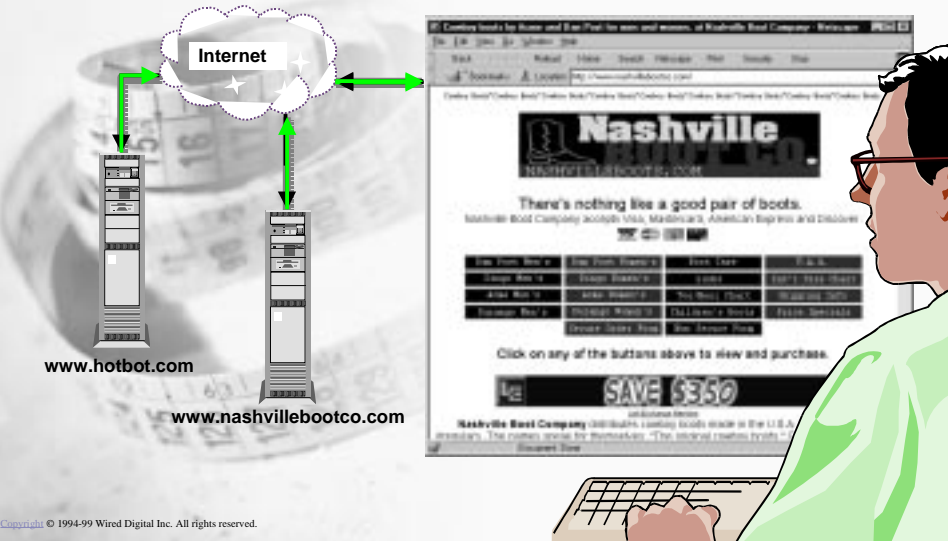
How to capture Business Processes



www.merc-int.com



What is a Business Process?



Internet

www.hotbot.com

www.nashvillebootco.com

Nashville
NASHVILLEBOOTCO.COM

There's nothing like a good pair of boots.
Nashville Boot Company accepts Visa, MasterCard, American Express and Discover.

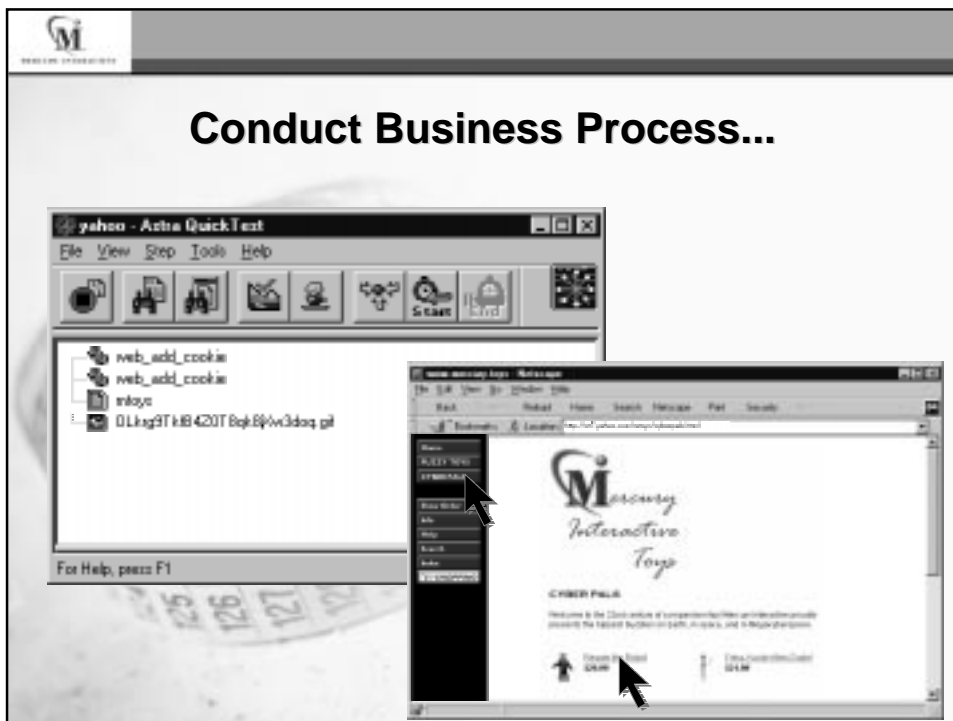
| | | | |
|------------------|------------------|-------------|-------------|
| Blue Suede Boots | Blue Suede Boots | Black Lace | Black Lace |
| Black Boots | Black Boots | Black Boots | Black Boots |
| Black Boots | Black Boots | Black Boots | Black Boots |
| Black Boots | Black Boots | Black Boots | Black Boots |

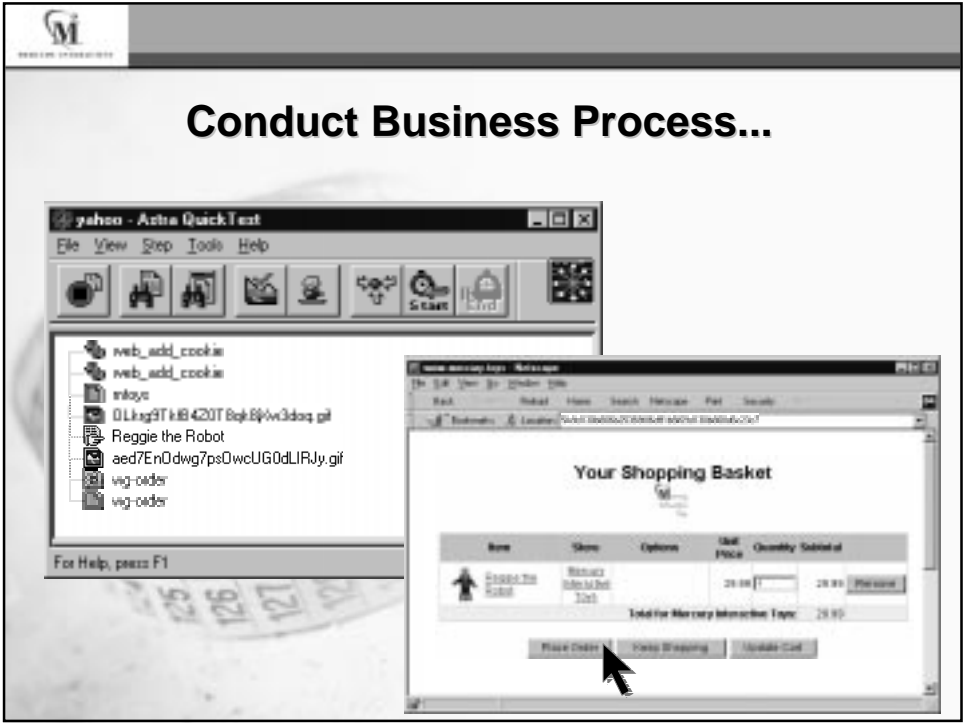
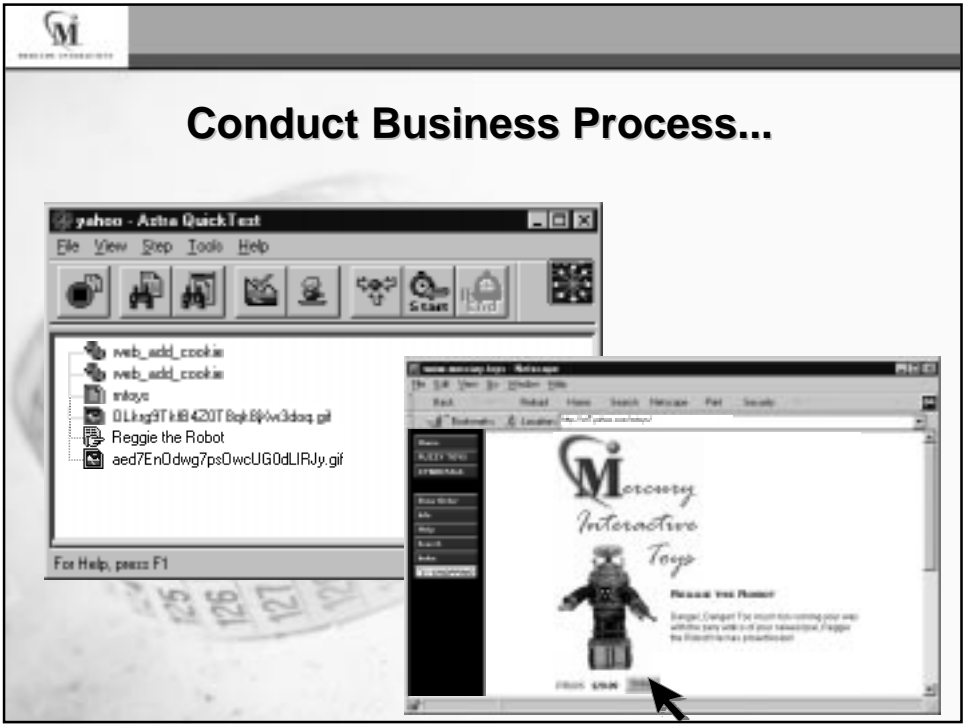
Click on any of the buttons above to view and purchase.

SAVE \$350

Nashville Boot Company distributes quality boots made in the U.S.A. members. The price shown for members is the lowest available price.

Copyright © 1994-99 Wired Digital Inc. All rights reserved.





Press Stop to end the Business Process Recording

The screenshot shows the 'yaho - Astro QuickTest' application. The 'File' menu is open, and the 'Stop' button (represented by a stop sign icon) is highlighted with a mouse cursor. The main window displays a list of test steps: 'Reggie the Robot', 'vg-order', 'wg-order', 'vg-final-order', and 'vg-confirm-order'. An inset window shows an 'Order Confirmation' page from 'Mango Interactive Test' with an order number of 8894-498. The page includes contact information for 'Mango' and a table of items.

| Item | Options | Unit Price | Quantity | Subtotal |
|--|---------|------------|----------|--------------|
| Product: The Robot | | 29.99 | 1 | 29.99 |
| Total for Mango Interactive Test: | | | | 29.99 |

Replay a Business Process

The screenshot shows the 'merctoys.ats - Astro QuickTest' application. The 'File' menu is open, and the 'Replay' button (represented by a play button icon) is highlighted with a mouse cursor. The main window displays a list of test steps: 'Reggie the Robot', 'vg-order', 'wg-order', 'vg-final-order', and 'vg-confirm-order'.



How to use Business Processes captures



www.merc-int.com



Functional Testing



www.merc-int.com



Automated Testing Benefits

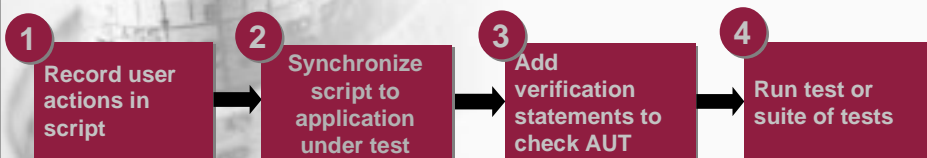
- Speed
- Repeatability
- Programming capabilities
- Coverage
- Reliability
- Reusability



Build 1 Build 2 Build 3... Build X



Automated Testing process



The screenshot shows a software testing tool interface. At the top, there is a menu bar with options: File, Edit, View, Record, Insert, Run, Debug, Tools, Help. Below the menu is a toolbar with various icons. On the left, a 'Browser' pane shows a tree view of test cases: "Mercury Tours", "Mercury Tours_2", "Mercury Tours_3", "info", "Mercury Tours_4", and "Mercury Tours_5". The "info" folder is expanded, showing sub-items: "depart" Select <Depart, "arrive" Select "New York", "seatPref_Aisle" Set "ON", and "findFlights" Click. The main area displays a browser window titled "Untitled Test" showing a Mercury Tours website. The website has a header with the text "Blue Skies, Smiling at Me" and a logo of a smiling sun with the text "mercury TOURS". Below the logo, there are two dropdown menus: "Departure City" with "Zurich" selected, and "Arrival City" with "Acapulco" selected. At the bottom of the interface, there is a 'Data Table' pane showing a table with columns A through H. The table has 6 rows. The first row is a header with 'Depart City' in column A. The second row has 'Frankfurt' in column A. The third row has 'New York' in column A. The fourth row has 'Sydney' in column A. The fifth row has 'Zurich' in column A. The sixth row is empty. The status bar at the bottom shows 'Ready' and 'CAP NUM SCRL'.

| | A | B | C | D | E | F | G | H |
|---|-----------|---|---|---|---|---|---|---|
| 1 | Frankfurt | | | | | | | |
| 2 | New York | | | | | | | |
| 3 | Sydney | | | | | | | |
| 4 | Zurich | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

What Is a Synchronization Point?

Definition:

A statement that instructs functional testing tool to wait for a certain response from the application during playback.

Establish Visual Cue to Synchronize
Examples

WINDOW
Wait for a window

OBJECT STATE
Wait for an object state

BITMAP
Wait for a bitmap to refresh

ELAPSED TIME
No visual cue: just wait for set time to elapse

What Is Verification?

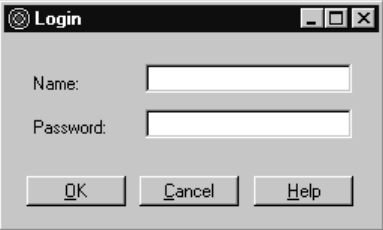
Definition:

Verify that the visual cues are appearing as expected and database operations are done as expected.

M
MERCURY INTERACTIVE

GUI Object Verification

- Check the state or attributes of GUI objects:
 - Is the window the correct size?
 - Is the OK button enabled?
 - What's the content of the Name field?




The screenshot shows a 'Login' dialog box with a title bar containing a maximize, minimize, and close button. The dialog has two text input fields labeled 'Name:' and 'Password:'. Below the fields are three buttons: 'OK', 'Cancel', and 'Help'.

M
MERCURY INTERACTIVE

Bitmap Image Verification

- Check non-GUI object areas of the application by capturing a bitmap
- Capture bitmap of window, object, or area of screen



The screenshot shows a web browser window displaying the Mercury Interactive website. The browser's address bar shows 'http://www.mercuryinteractive.com/'. The website header includes the Mercury Interactive logo, which is highlighted with a red box. The logo consists of a stylized 'M' with a globe inside. The website content includes the text 'MERCURY INTERACTIVE' and 'AUTOMATED SOFTWARE TESTING SOLUTIONS'. A navigation menu on the right side lists 'e-business', 'erp', 'year 2000', 'client/server', and 'euro'.



MOORE INVESTMENTS

Text Verification

- Read and verify text from bitmap areas or non-GUI objects based interfaces (e.g., ASCII)

| Investment Option by Asset Class | Balance | Shares or Units | NAV | \$ Change per Share or Unit |
|-------------------------------------|----------|--------------------|---------|--------------------------------|
| Stock Investments | | | | |
| International Stock | | | | |
| <u>DIVERSIFIED INTERN'L</u> | \$536.88 | 31.287 | \$17.16 | -\$0.17 |
| Mid/Large Cap Stock | | | | |
| <u>MAS VALUE ADVISER</u> | \$559.64 | 32.594 | \$17.17 | -\$0.08 |
| Blended Investments | | | | |
| <u>PURITAN</u> | \$266.75 | 13.893 | \$19.20 | -\$0.01 |



MOORE INVESTMENTS

Database Verification

- Compares database values with end-user input to ensure transaction accuracy.

| Expected Data | | | Actual Data | | | | |
|---------------|---------------|-----------|--------------|-----------|---------------|------|----|
| Customer No. | Share No. | Order No. | Customer No. | Share No. | Order No. | | |
| 1 | Stock Advisor | 5218 | 9 | 1 | Stock Advisor | 5218 | 9 |
| 2 | Jack Smith | 4296 | 2 | 2 | Jack Smith | 4296 | 2 |
| 3 | Jack Blowers | 5232 | 3 | 3 | Jack Blowers | 5232 | 3 |
| 4 | James Day | 3291 | 7 | 4 | James Day | 3291 | 7 |
| 5 | Auto Haines | 4214 | 10 | 5 | Auto Haines | 4214 | 10 |
| 6 | Ann Skane | 4216 | 5 | 6 | Ann Skane | 4216 | 5 |
| 7 | Ann Smith | 1158 | 13 | 7 | Ann Smith | 1158 | 13 |
| 8 | Ann Smith | 1159 | 14 | 8 | Ann Smith | 1159 | 14 |
| 9 | Ann Smith | 1159 | 15 | 9 | Ann Smith | 1159 | 15 |
| 10 | Ann Smith | 5230 | 1 | 10 | Ann Smith | 5230 | 1 |
| 11 | Ann Smith | 4216 | 6 | 11 | Ann Smith | 4216 | 6 |
| 12 | Ann Smith | 1158 | 13 | 12 | Ann Smith | 1158 | 13 |
| 13 | Ann Smith | 5230 | 1 | 13 | Ann Smith | 1159 | 14 |
| 14 | Ann Smith | 4216 | 6 | 14 | Ann Smith | 4216 | 6 |
| 15 | | | 15 | | | | |

Row 1 of 15: Expected [Customer Name] = John Day, Actual [Customer Name] = John Day
 Row 2 of 15: Expected [Customer Name] = Jonathan, Actual [Customer Name] = Jonathan
 Row 3 of 15: Expected [Customer Name] = Van Smith, Actual [Customer Name] = Van Smith
 Row 4 of 15: Expected [Customer Name] = Mary Smith, Actual [Customer Name] = Mary Smith

View the Execution Summary

The screenshot shows the 'Astra QuickTest Result Summary' window. The main content area displays the test name 'merc_toys' and the execution time 'Executed: 2/7/99 5:40:19 PM'. Below this is a table with columns for 'Run #', 'Summary', and three 'Service:' entries. The first row shows a successful run with checkmarks in the Summary and Service columns.

Annotations include a callout for the 'Report Tree pane' on the left and another for the 'Report Details pane' at the bottom right of the window.

To the right of the window is a file tree diagram showing a folder 'merc_toys' containing a sub-folder 'result 1', which in turn contains a file 'merc_toys.rtp'. Below the diagram is the label 'Results file'.

Functional Testing Solutions

Open Test Architecture : Integrate Other Application Lifecycle Mgmt Tools


| | | |
|---|--|--|
| <p>Web/e-business</p> <p><i>Astra QuickTest</i> for: Fast, simple web testing</p> <p><i>WinRunner</i> for: Testing Enterprise Web-based Applications</p> | <p>Packaged Applications</p> <p><i>WinRunner</i> for: Baan SAP Oracle Applications Peoplesoft</p> | <p>Y2K/Euro</p> <p><i>WinRunner</i> for: Y2K / Euro Testing</p> |
| <p>Custom Client/Server</p> <p>Regression & Functional Testing: <i>WinRunner</i></p> | | |
| <p>Test Management: <i>TestDirector</i></p> | | |



Load Testing Testing

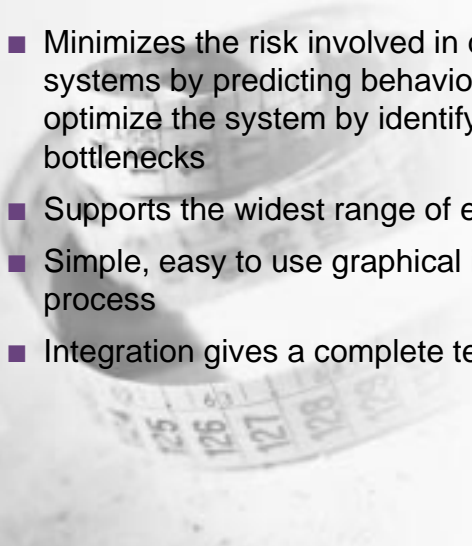


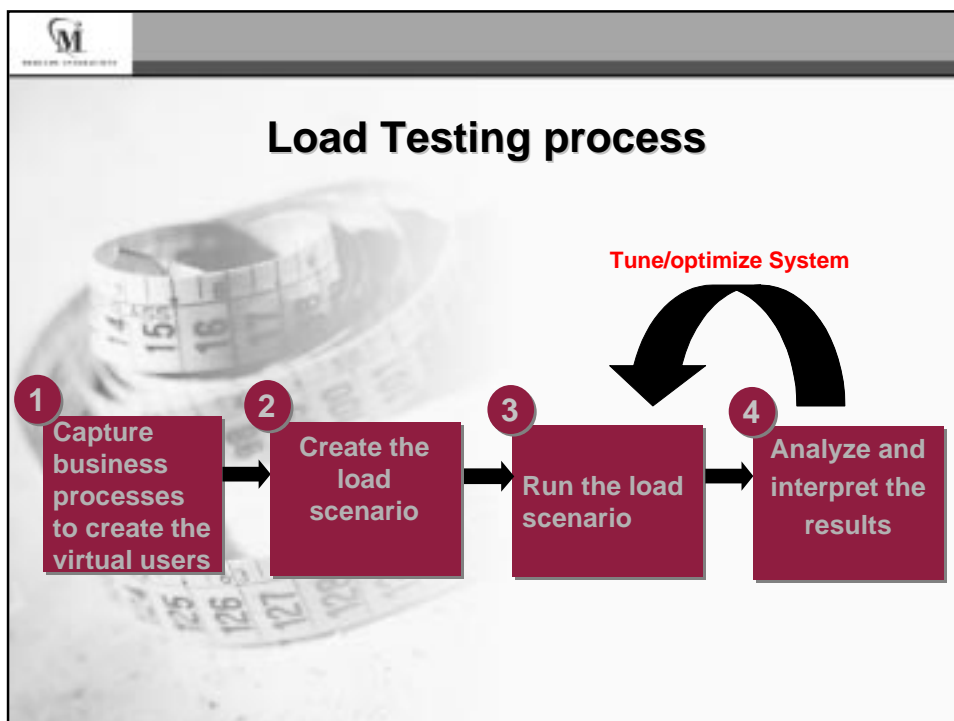
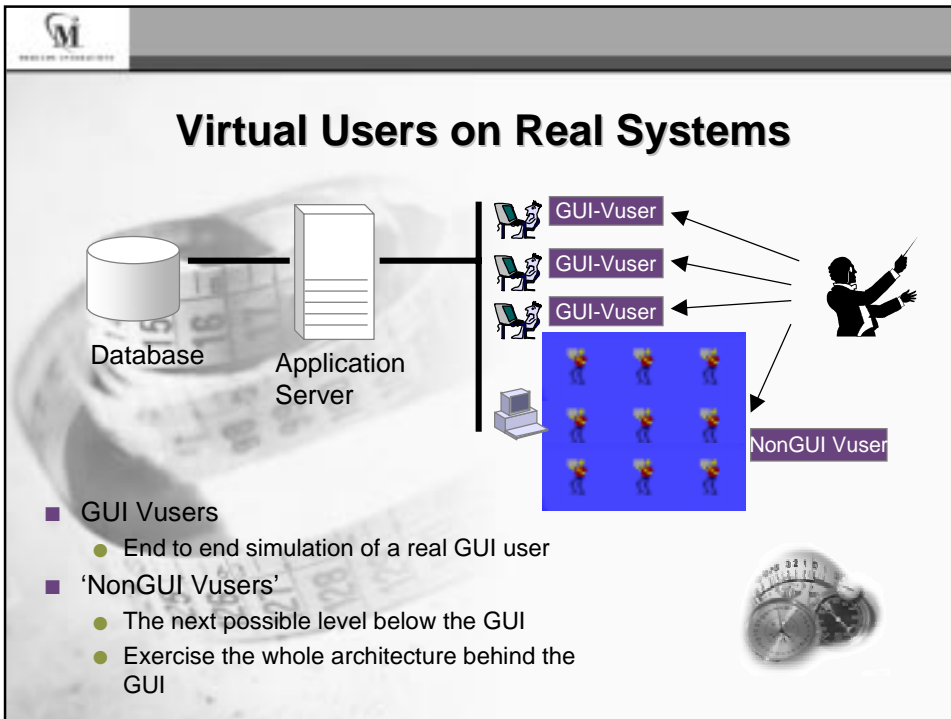
www.merc-it.com



Benefits

- Minimizes the risk involved in deploying complex enterprise systems by predicting behavior and performance. It helps optimize the system by identifying and isolating performance bottlenecks
- Supports the widest range of environments
- Simple, easy to use graphical interface accelerates testing process
- Integration gives a complete testing solution





Definition: Transaction

An end-to-end measurement of one or more user actions within a recorded business process.

Input Order Information Measure Save Order

Measure Entire Update Order

Setting up Transactions –Example

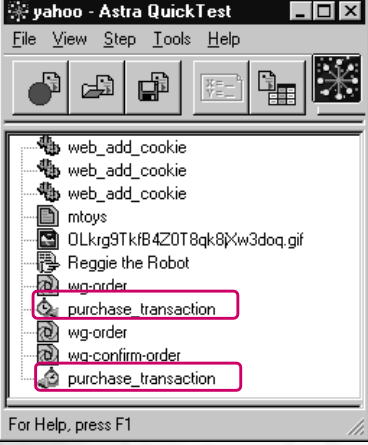
Transaction 1 Measure Logon process

Transaction 2 Overall measurement of all non-logon processes

Transaction 3 Measure a database-intensive process within main transaction

- www.mtoys.com
- Logon
- web_add_cookie
- web_add_cookie
- mtoys
- OLkrg9TkfB4Z0T8qk8Xw3doq.gif
- Reggie the Robot
- wg-order
- wg-confirm-order

The Start and Stop Transaction Icons

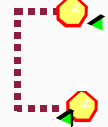


The screenshot shows the Astra QuickTest interface with a list of transactions. The 'purchase_transaction' entries are highlighted with pink boxes. The list includes: web_add_cookie, mtoys, DLkrg9TkfB4Z0T8qk8pXw3doq.gif, Reggie the Robot, wg-order, purchase_transaction, wg-order, wg-confirm-order, and purchase_transaction.

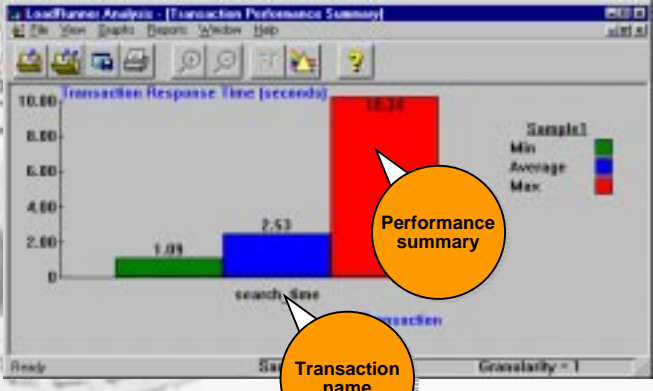
Common icons and usage examples

Start Transaction
Set starting point for measurement

End Transaction
Set end point for measurement



Transaction Analysis – Example



The screenshot displays a bar chart titled 'Transaction Response Time (seconds)' for the 'search_time' transaction. The chart shows three bars representing performance metrics: Min (1.09), Average (2.53), and Max (18.38). A legend on the right indicates that the bars represent Sample 1, Min, Average, and Max. Two callout boxes are present: one pointing to the Max bar labeled 'Performance summary' and another pointing to the transaction name on the x-axis labeled 'Transaction name'.

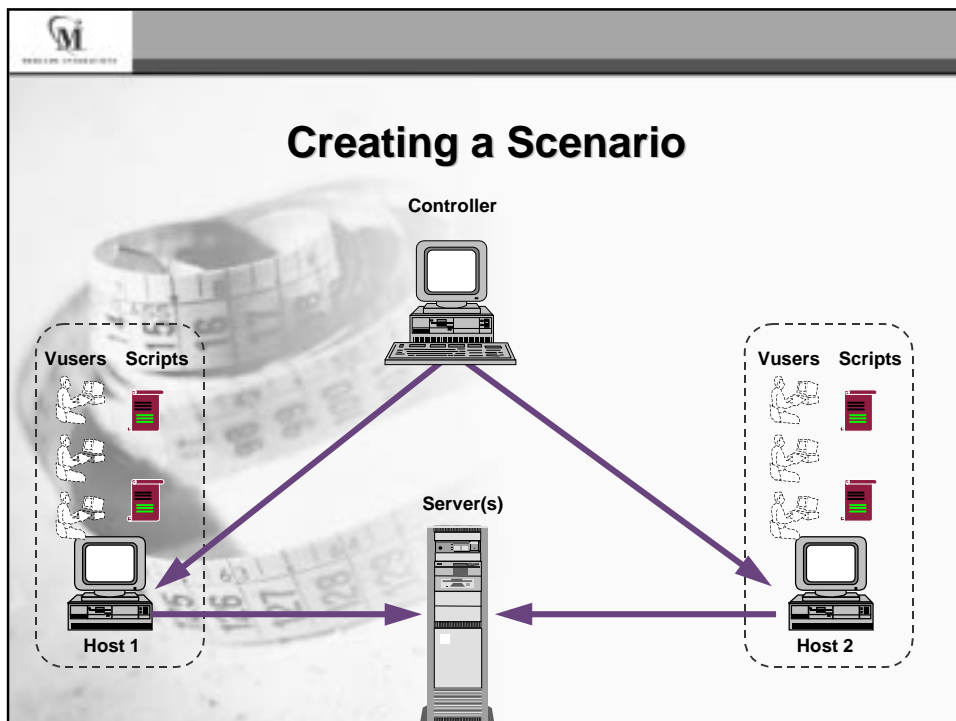
| Metric | Value (seconds) |
|---------|-----------------|
| Min | 1.09 |
| Average | 2.53 |
| Max | 18.38 |

Definition: Scenario

The events and conditions that define a load test, such as:

- Number of Vusers
- Vuser actions (scripts)
- Machines that Vusers run on(hosts)

The diagram consists of four interlocking puzzle pieces arranged in a square. The top-left piece is labeled 'virtual users', the top-right 'host machines', the bottom-left 'results location', and the bottom-right 'load conditions'. In the center, where all four pieces meet, is a smaller piece labeled 'test scripts'.



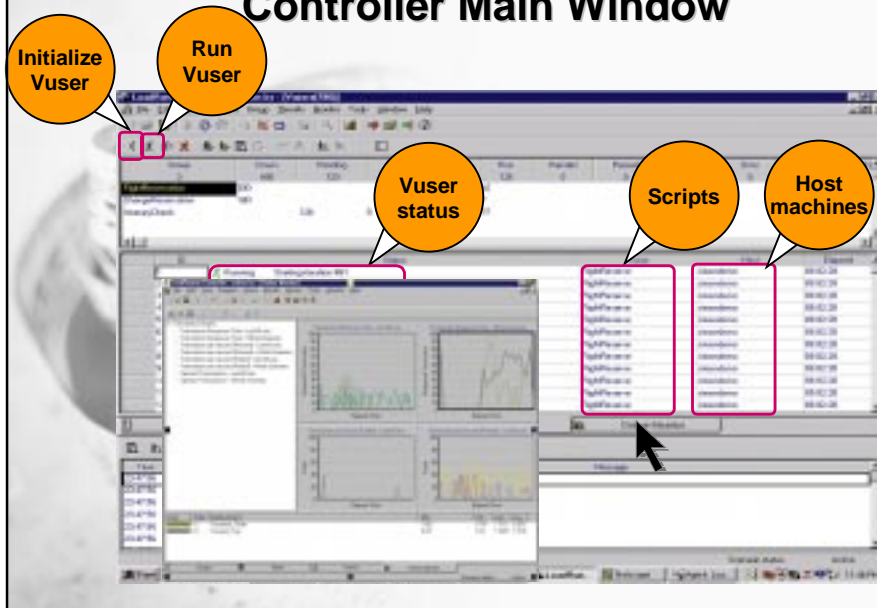


What the Controller Provides

- Before scenario execution
 - creates scenario
 - sets up run-time configuration
- During scenario execution
 - runs many Vusers simultaneously
 - controls each Vuser (load, run, pause, stop)
 - displays execution status of each Vuser
 - displays execution messages from each Vuser
 - collects performance data
- After scenario execution
 - launches the Analysis tools
 - analyzes system performance



Controller Main Window



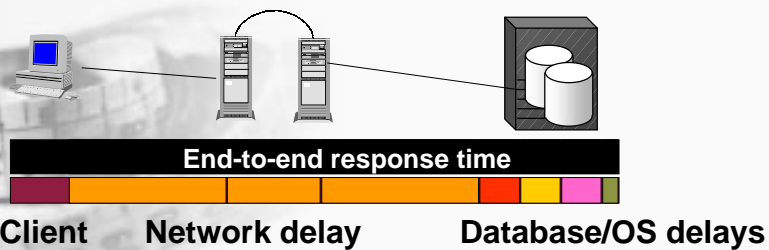


What MI's Load Testing Tools Provide

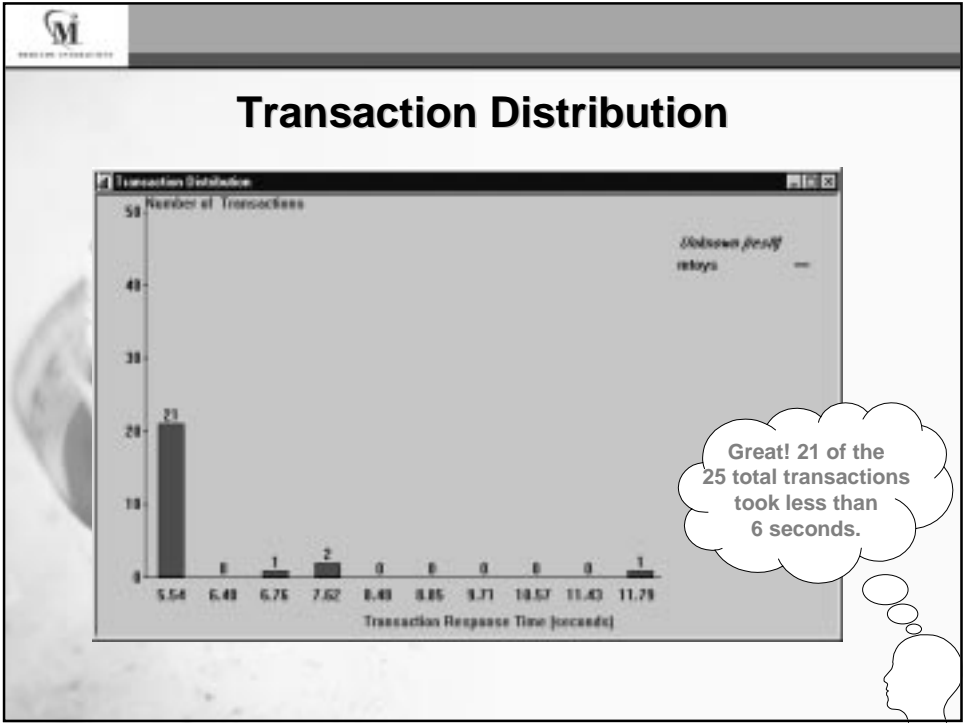
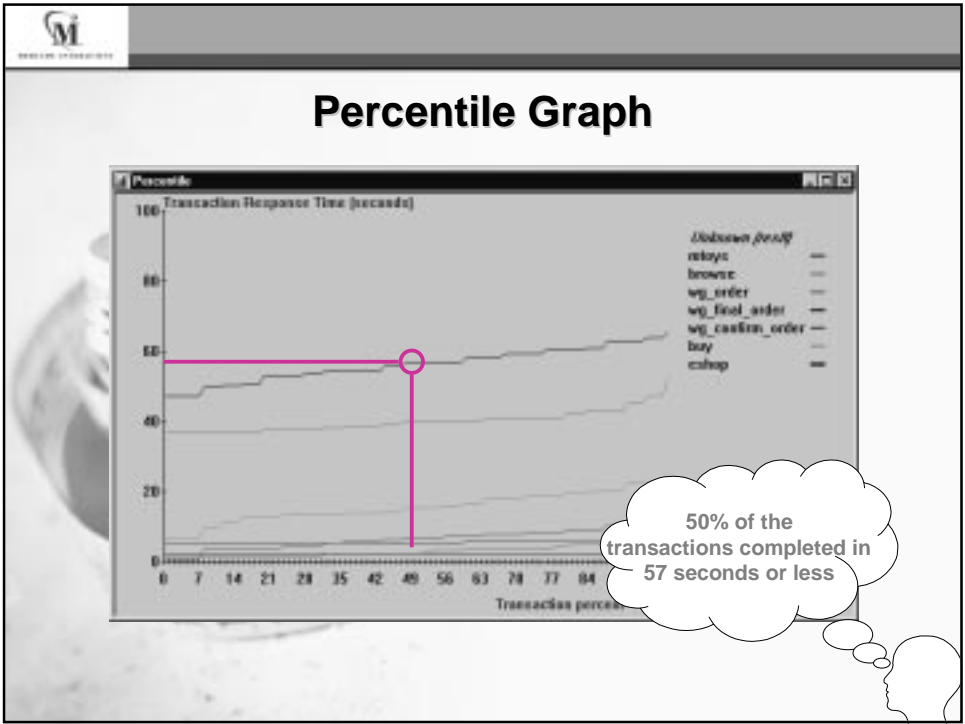
- Automatic Programmatic IP-Multicasting
 - feature that emulates different IP addresses for virtual users
- Allowing users to :
 - Exercise network devices (hubs, bridges, routers)
 - Test firewall security impact on performance
- Ability to handle 50,000+ virtual users
- Scheduler
 - allows automatic ramp up of virtual users during a scenario
- Web performance monitors
 - that measures & displays transaction time spent on Web server, network, & app server

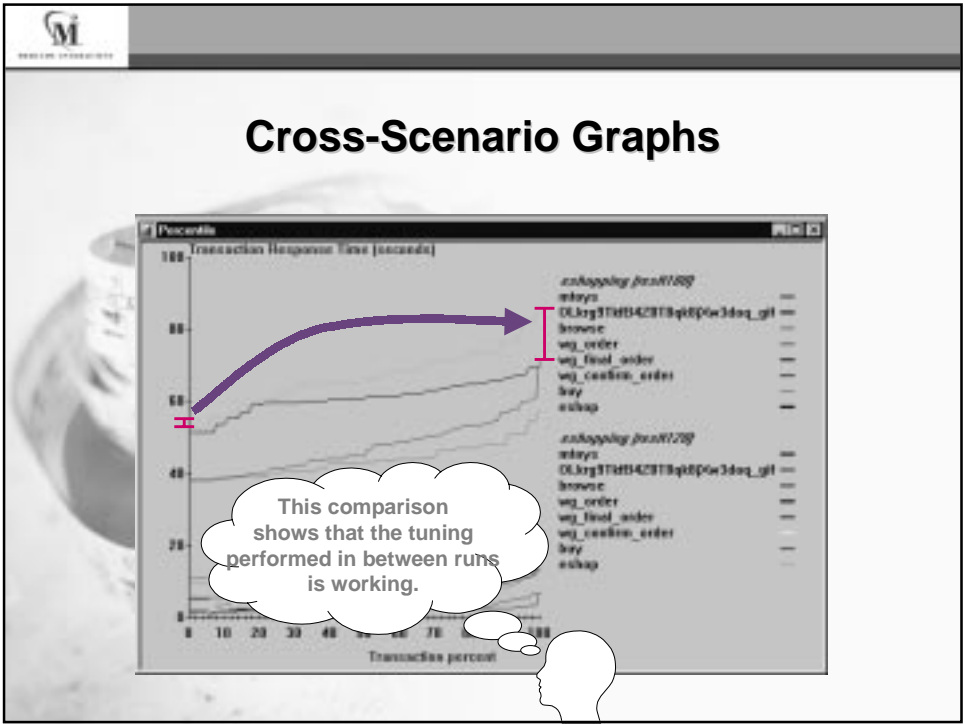


Isolates bottlenecks



- Collects and presents data from client, network, & server
 - Transaction Breakdown Monitor, Network Delay Monitor, Server Monitor
 - SNMP Monitor, Tuxedo Monitor






Transaction Performance Summary Report

Scenario: *Default*
 Results: *e:\scenario\10\transaction*
 Start Time: 10 - Feb - 08 13:00:27
 End Time: 10 - Feb - 08 18:00:11
 Duration: 00:04:43.880

| Transaction | Pass | Fail | Min | Avg | Max | 90th Percentile | AvgStdDev |
|------------------------------------|------------|----------|-------|-------|--------|-----------------|-----------|
| <i>buy</i> | 25 | 0 | 5.25 | 9.05 | 12.50 | 9.24 | 1.50 |
| <i>DL.org37fd04c2f8q0pw3daq_gi</i> | 25 | 0 | 5.50 | 6.50 | 10.00 | 6.50 | 0.90 |
| <i>buy_confirm_order</i> | 25 | 0 | 6.25 | 6.40 | 9.00 | 6.30 | 0.80 |
| <i>buy_order</i> | 25 | 0 | 6.00 | 7.00 | 9.75 | 6.80 | 0.90 |
| <i>shopping_test100</i> | 25 | 0 | 7.14 | 5.20 | 14.00 | 10.00 | 2.00 |
| <i>buy_final_order</i> | 25 | 0 | 5.00 | 6.50 | 9.00 | 7.00 | 1.10 |
| <i>browse</i> | 25 | 0 | 3.00 | 3.00 | 9.00 | 6.70 | 1.00 |
| <i>nlays</i> | 25 | 0 | 20.00 | 40.00 | 200.00 | 40.00 | 3.40 |
| <i>buy_order</i> | 25 | 0 | 1.90 | 3.80 | 9.00 | 3.80 | 2.00 |
| <i>buy_final_order</i> | 25 | 0 | 2.00 | 2.40 | 3.00 | 2.20 | 0.40 |
| <i>buy_confirm_order</i> | 25 | 0 | 0.70 | 6.00 | 11.00 | 10.00 | 2.00 |
| <i>buy</i> | 25 | 0 | 7.00 | 16.00 | 26.00 | 20.00 | 6.00 |
| <i>nlays</i> | 25 | 0 | 47.10 | 30.00 | 40.00 | 40.00 | 4.00 |
| Total: | 250 | 0 | | | | | |

(*) Failures - does not include the recorded transactions

 **Load Testing Solutions**

Open Test Architecture : Integrate Other Application Lifecycle Mgmt Tools

| | | |
|---|---|---|
| <p>Web/e-business</p> <p><u>Astra LoadTest</u> for: Fast, simple web testing</p> <p><u>LoadRunner</u> for: Testing Enterprise Web-based Applications</p> | <p>Packaged Applications</p> <p><u>LoadRunner</u> for: Baan SAP Oracle Applications Peoplesoft</p> | <p>Y2K/Euro</p> <p><u>LoadRunner</u> for: Y2K / Euro Testing</p> |
| <p>Custom Client/Server</p> <p>Load and Performance Testing: <u>LoadRunner</u></p> | | |
| <p>Test Management: <u>TestDirector</u></p> | | |

 **Application Performance Management**



www.merc-int.com



Application Performance Management Today

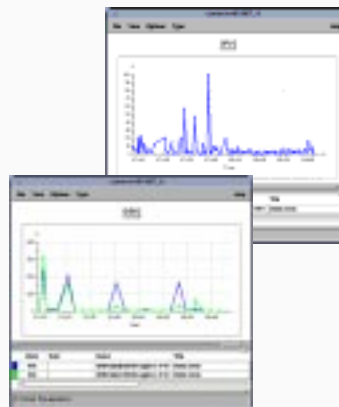
- Performance is not measured at the business process level
 - Various “System” monitors generate confusing data
- Application availability data is not meaningful to end-users
 - Reports are difficult to extrapolate outside of IT
- IT is unable to anticipate application disruptions
 - IT is forced to be reactive to user demands



Current Solutions

Existing tools are net/server centric

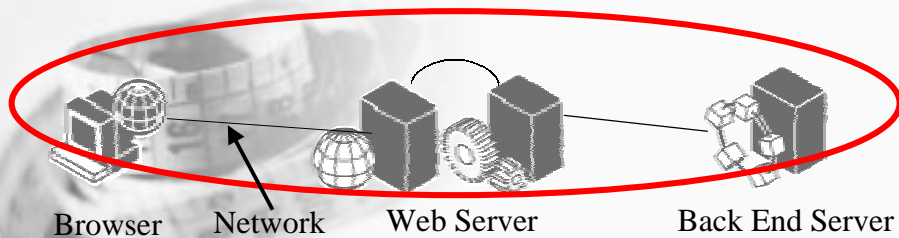
- Total requests made per hour/done/outstanding
- Status of each server
- Detailed report on the domain configuration
- Boot and shut down by domain, machine, group and server
- Monitors status of the Process Scheduler
- Monitors status of jobs queued through the process scheduler
- Tracks Process Server statistics



Lots of Data, No End User Perspective



Introducing: “Topaz”

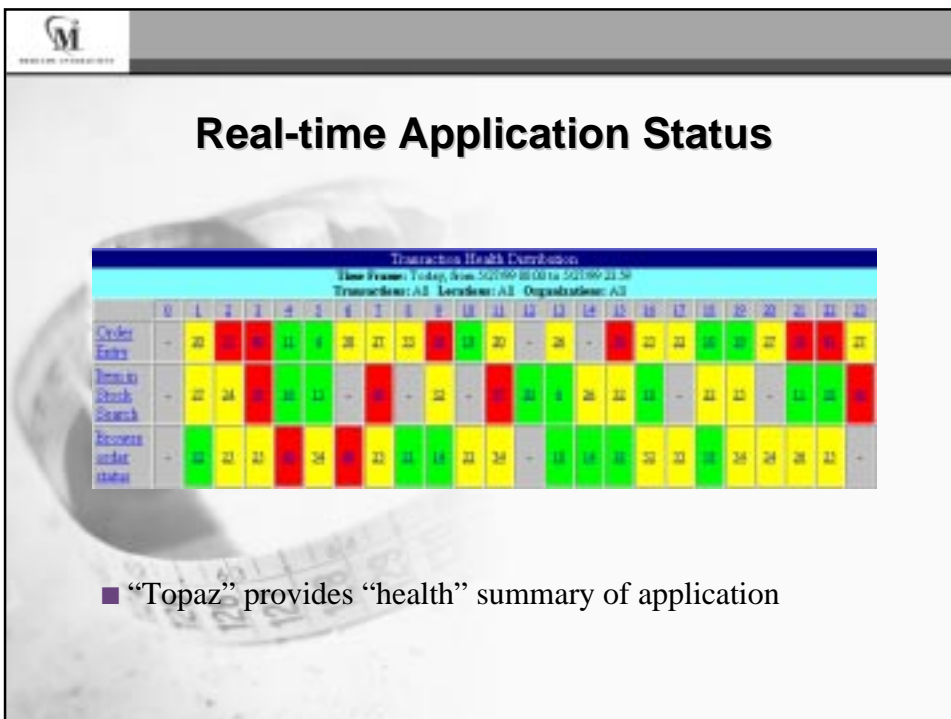
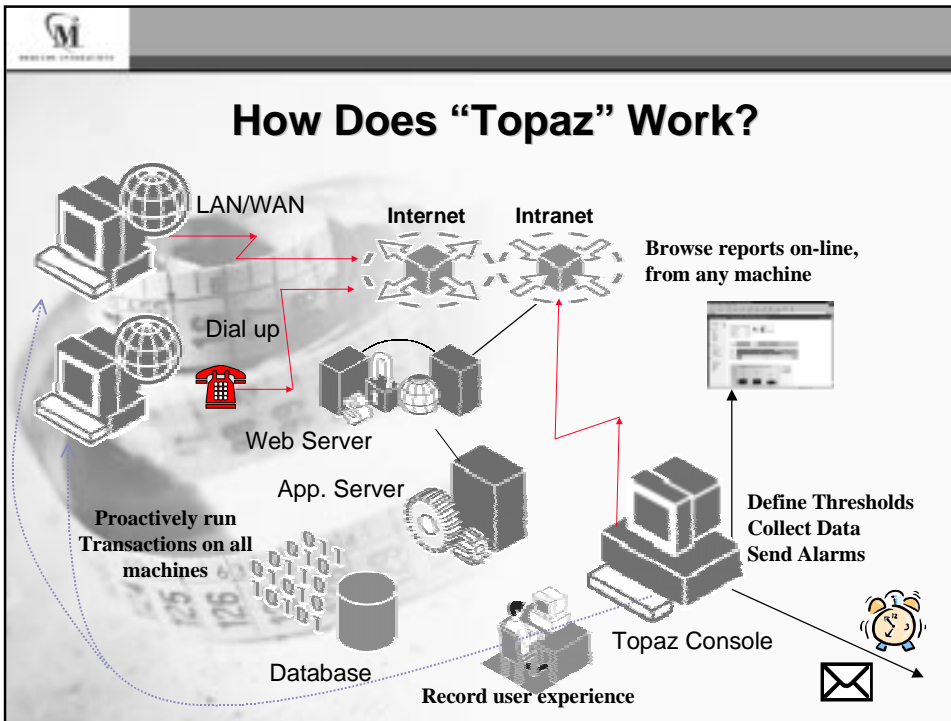


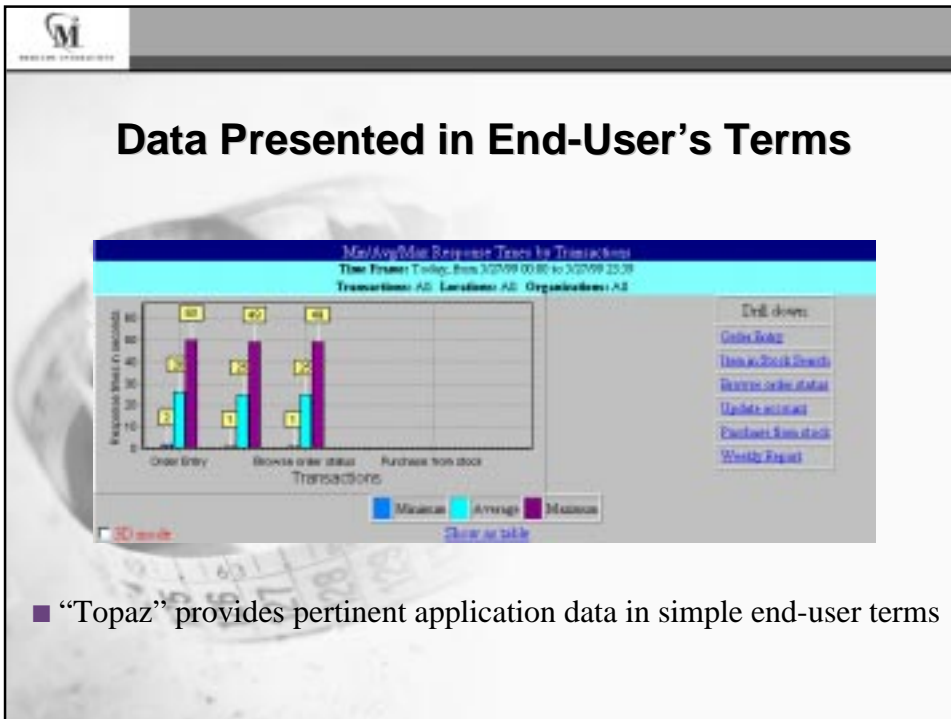
- “Topaz” measures end-user experience
 - Client perspective
 - Proactively measures actual business processes as opposed to system data
 - Provides meaningful data on application performance and availability



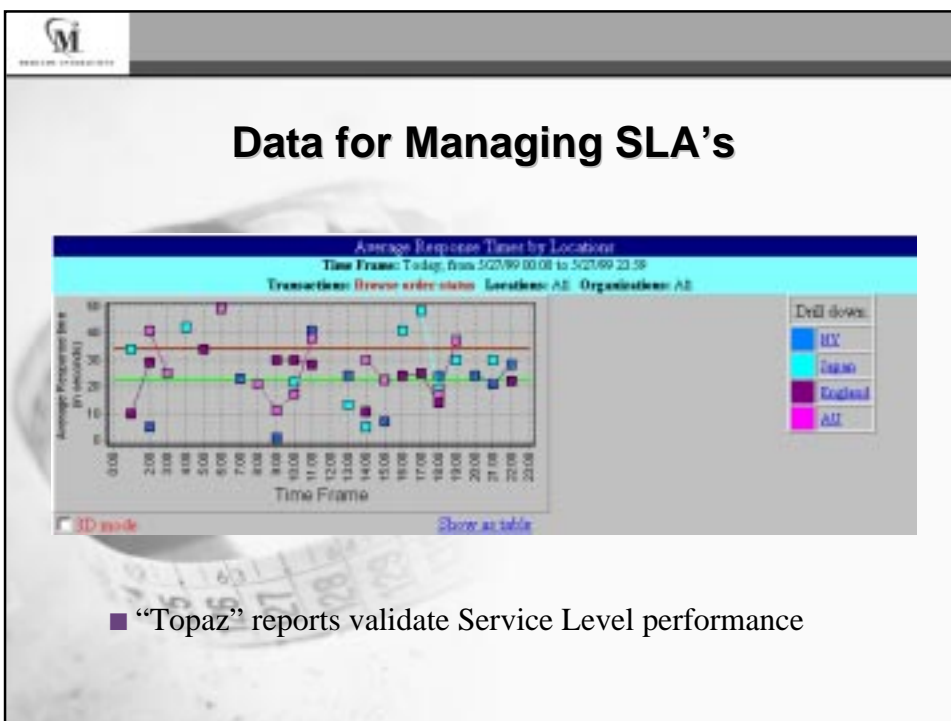
“Topaz” Benefits

- Bridges gap between business users and IT organizations
 - Talks the language of the user - Transactions & Business Processes
- Delivers quantifiable, measurable user experience to optimize application and infrastructure
 - Define and enforce Service Level Agreements
 - Maintain quality control of outsourcing agreements
- Helps identify performance problems **proactively**
 - Alert business user to application problems, before they find them





- “Topaz” provides pertinent application data in simple end-user terms



- “Topaz” reports validate Service Level performance



HERCURY INTERACTIVE

“Topaz” Capabilities

- Measuring & reporting End-user experience
 - Availability, Performance.
- Drive simulated users, replacing the real users
- On-line monitoring of the end-user experience
- Real-time alerts (E-mail, pager) based on thresholds
- "Anytime, Anywhere" monitoring - browser based SLM graphs and reports



HERCURY INTERACTIVE

Mercury Interactive Background



www.merc-int.com



Mercury Interactive Worldwide

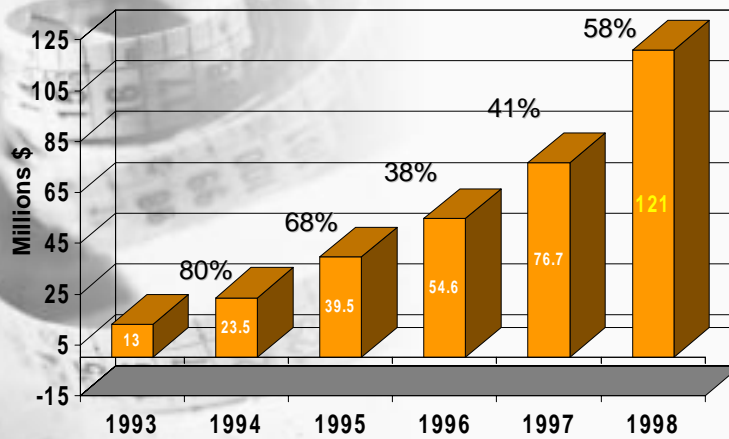


- ◆ **Founded in August 1989**
- ◆ **IPO in October 1993 -- NASDAQ : MERQ**
- ◆ **Headquarter in Sunnyvale, Silicon Valley**
- ◆ **600+ employees in 30 worldwide locations**
- ◆ **Regional HQ in München, Paris, London, Brussels, Stockholm**



Mercury Interactive : Accelerating Growth

■ \$21.8m Net Profit in 1998

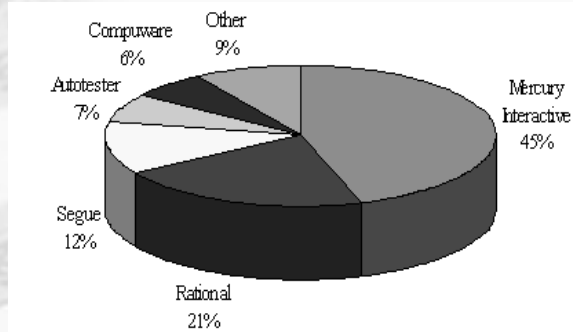




Mercury Interactive : Undisputed Dominant Leadership

**Independent Analysts confirm
Mercury Interactive's leading position:**

Forrester
Gartner Group
Giga
IDC
Meta Group
Yankee Group



Source: the Yankee Group, 1998



The Industry Confirms : MI's Thought Leadership



"Mercury Interactive continues to raise the bar in delivering automated testing tools...it succeeds in delivering depth of capabilities across a breadth of custom, packaged application, Internet and Year 2000 and Euro testing environments."



STAR Award-Winning Customer Support
Recognized for excellence in Electronic Support
Only Testing Tool company ever recognized



The Case for Auditing Year 2000 Software Repairs

**How to Mitigate the Risks of Computer Failures and Reduce the
Cost of Year 2000 Compliance**

Version 3.0

Table of Contents

| | |
|--|-----------|
| 1. Introduction | 3 |
| 2. Why Audits of Year 2000 Software Repairs Are Necessary | 5 |
| 3. The Benefits of Software Inspections | 7 |
| 4. How Inspections Lowers Testing Risks and Costs..... | 8 |
| 5. How to Perform an Inspection | 11 |
| Who Should Perform an Inspection | 12 |
| 6. Inspection Scenarios..... | 12 |
| Internally Remediated Applications | 12 |
| Offsite or Offshore Factory-Remediated Code..... | 13 |
| Consulting Project Audits | 13 |
| Independent Validation and Verification (IV&V) | 13 |
| Vendor Software Packages..... | 13 |
| Key Supplier Applications..... | 14 |
| Maintaining Compliance After Remediation | 14 |
| 7. Inspection Case Studies | 15 |
| Telecommunications Company #1 | 15 |
| Telecommunications Company #2 | 15 |
| Utility Company | 15 |
| Aerospace Company..... | 16 |
| Health Insurance Company..... | 16 |
| 8. Conclusion..... | 16 |
| 9. Footnotes | 17 |
| 10. About Reasoning, Inc..... | 18 |

THE CASE FOR AUDITING YEAR 2000 SOFTWARE REPAIRS

HOW TO MITIGATE THE RISKS OF COMPUTER FAILURES AND REDUCE THE COST OF YEAR 2000 COMPLIANCE

1. Introduction

Unprecedented information technology (IT) resources are now being devoted to resolving the Year 2000 software crisis. It is a profound business problem with a fast approaching immovable deadline. Many companies still face a staggering amount of software repair and testing work with too few resources.

The Year 2000 problem is an extreme case of software maintenance where thousands of extra bugs per million lines of code will have to be identified, repaired, and tested. When software applications require such extensive debugging and modification, even the most skilled programmers under the best of circumstances cannot find all the bugs and invariably introduce new bugs. As we enter the next century, somehow all fixes must be completed correctly and validated to ensure that business operations are uninterrupted.

Business disruptions caused by faulty software can be costly and sometimes devastating in terms of lost business, damage to customers, and declines in stock value, which in turn may lead to litigation. Reports of serious disruptions caused by software bugs are commonplace. The following recent examples were found in *The New York Times* online news:

- A large telecommunications company's 800 services outage: A 1.5-hour outage caused by a software problem cost customers several hundred million dollars.
- New Jersey Department of Motor Vehicles computer problem: A new software program failed in the first hour after installation, forcing all 45 field offices to turn away thousands seeking licenses, registrations, and other services.
- Oxford Health Plans billing problem: Billing and payment software problems resulted in failure to collect hundreds of millions of dollars from member hospitals and doctors, causing large losses and a drop in Oxford's market value of \$3 billion in one day.
- Union Pacific tracking problems: Software problems resulted in massive rail tie-ups, costing businesses billions of dollars.
- Snap-On, Inc. order processing problem: Software bugs in a new order entry system cost the company \$50 million in sales for the first half of 1998 and caused second quarter earnings to drop 40 percent compared with the same period in 1997.

Due to the pervasiveness and complexity of Year 2000 software bugs, experts believe that serious business disruptions will escalate exponentially. Every company needs to ask itself whether it has done enough to ensure that operations will continue into the next century and, if so, whether the company can withstand the anticipated onslaught of litigation and survive unscathed.

This paper explains how audits of remediated software offer companies a strategic line of defense against Year 2000 risks from a business operability and legal protection perspective. The main benefit of auditing software repairs is that it dramatically increases the chances of finding fatal errors that have been omitted, introduced, or fixed incorrectly during the Year 2000 remediation process. Section 2 explains why traditional software testing generally regarded as the final software quality safety-net doesn't ensure all serious errors will be found in a Year 2000 scenario. It is well documented that standard testing misses many errors and in certain circumstances, inadequate testing is inevitable because of insufficient test data, testing skills, subject matter expertise, cost, and lack of time.

Inadequate Year 2000 testing in particular may expose a company to serious legal liabilities and compromise a successful legal defense by serving to indict a company for reckless or negligent behavior in the face of a known risk. The legal community has been mobilizing to deal with Year 2000-related litigation and in fact several cases have already been filed and the case load is starting to build. Robert T. Russell, an attorney specializing in Year 2000 insurance matters for the Silicon Valley law firm of Gray, Carey, Ware & Freidenrich, warns that a wave of Year 2000-related lawsuits are inevitable and that companies should not assume existing business insurance protects the company or individual officers from liability. According to Mr. Russell, common property/business interruption and commercial general liability policies are unlikely to cover the more common types of Year 2000 claims while certain forms of errors and omissions and directors and officers liability policies may offer limited protection. Consequently, attorneys and risk management consultants advise corporate clients to use software audits as a means of enhancing a company's due diligence file to support the use of the "business judgement rule" as a defense against Year 2000-related shareholder actions and negligence law suits.

2. Why Audits of Year 2000 Software Repairs Are Necessary

In an ideal world, where all software development and maintenance are properly done the first time, there is no reason to conduct testing to validate the functionality and performance of software. Unfortunately, errors in software are inevitable in the real world. The complexity and breadth of Year 2000 compliance issues in application software, processes, and embedded systems make it impossible for any analysis or remediation method to identify and fix all possible problems. The software testing process has traditionally served as the last line of defense, protecting business operations from disastrous failures and legal liabilities. That is why testing has been designated as the most critical phase of a Year 2000 compliance effort.

The primary goal of testing is to find and eliminate as many errors as possible. However, based on the recent findings of several industry authorities and real-world case studies (see Section 7), it is evident that traditional testing practices are often inadequate for verifying Year 2000 compliance, and that audits of Year 2000 software repairs are necessary for many reasons:

- Testing misses many Year 2000 software defects.
- New errors are introduced during the remediation process. Leading Year 2000 consultants, Ian Hayes and William Ulrich state in their book, The Year 2000 Software Crisis - The Continuing Challenge, that programmers, regardless of skill level, on average introduce three new errors for every 100 changes they make to the code.
- The quality of Year 2000 software repairs performed by external service providers is suspect. According to the Gartner Group, 90 percent of the remediation jobs performed by external service providers contain quality problems.
- A high percentage of applications repaired internally contain fatal errors and inconsistent repairs. A recent survey conducted by the Information Technology Association of America found that 44 percent of the respondents experienced Year 2000 failures after remediation under operating conditions and 67 percent reported failures during tests.
- Compliant applications put back into production have been “re-contaminated” through routine maintenance procedures or interaction with noncompliant systems. A survey conducted in March 1998 by Market Perspectives Inc., reported that 12 percent of the 300 companies surveyed reported Year 2000 re-contamination problems.
- Vendor packages and supply chain systems are not Year 2000 compliant.

This data begs the question of why software testing fails to identify a significant number of Year 2000 bugs and is an insufficient means of ensuring compliance. The answer is that Year 2000 testing poses the following unique challenges, different from standard software testing.

- **Fixed timeframe.** The end date of a Year 2000 project cannot be postponed. Many applications will encounter date-related windows of failure even before the century transition. Adequate testing must be completed before these event horizons.
- **Inadequate test environments.** Year 2000 testing requires specialized tools and techniques including the ability to simulate future-dated environments, manipulate test data and perform intelligent comparisons. Executing a large volume of concurrent tests also requires additional hardware capacity.
- **Lack of testing skills.** While most programmers have experience in performing maintenance changes and generating simple test cases to exercise those changes, Year 2000 testing requires far more extensive testing skills. A skilled Year 2000 tester must be able to create and manipulate test data, conduct rigorous tests of all types (regression, future date, boundary) and at all levels (unit, integration, system), and to perform test coverage analysis.
- **Lack of subject matter expertise.** Few Year 2000 testing teams have the luxury of fully understanding the applications they are testing. Few applications have adequate documentation, and in most cases, the original experts have long since moved to other jobs. Application expertise enables testers to make informed decisions on where best to apply their efforts. Without this experience, testers are forced to rely on the adequacy of their testing techniques and the depth of their test coverage.
- **Far more changes to test.** In a single application of one million lines of code, 2,000 to 3,000 or more Year 2000-related modifications are typically required. This is a far greater number of changes than required during routine maintenance, and the changes span multiple programs and databases. These changes also tend to be scattered throughout the code, making it difficult for programmers to consistently implement fixes.
- **Poor test data.** Most applications have little, if any, test data. Lack of test data can lead to low levels of test coverage, which can easily allow defects to slip through testing into production. Many testers plan to rely on subsets of production data as the basis for their tests. Production data rarely covers more than 20 to 30 percent of the paths in an application.
- **Insufficient resources.** Companies must test almost every application, operating system, platform, package, and environment for Year 2000 compliance. With little time remaining, hundreds of concurrent test projects will compete for the same hardware and human resources.

Because basic testing already accounts for 50 percent or more of the average Year 2000 project resources and costs, extreme testing is not feasible in most cases. Compounding the problem, recent industry statistics show that remediation projects are behind schedule since many organizations have miscalculated the complexity, time, and resources required to implement repairs. For example, a survey published by CapGemini on July 20, 1998, found that the percentage of companies missing Year 2000 project milestones from December 1997 through April 1998 rose from 78 percent to 84 percent. And according to the META Group, a prominent IT advisory firm in Stamford, Connecticut, more users are falling behind in Year 2000 remediation (*META FAX July 13, 1998*). To make matters worse, a recent Gartner Group survey of 6,000 companies worldwide found that **more than 50 percent said they are not planning to do any Year 2000 compliance testing** because of lack of time and resources. Testing can get even more complicated for companies that must also perform tests on packaged software and the interdependent software programs used by key suppliers. All this means that many organizations will have to cut short the time available for basic testing, as illustrated in Figure 1.

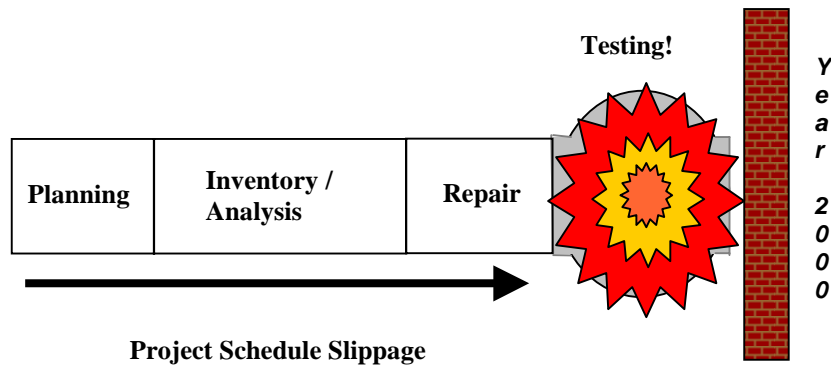


Figure 1: Testing is constrained by Year 2000 repair overruns.

3. The Benefits of Software Inspections

Given the limitations of testing, companies must seek other means to compensate and increase the odds of catching Year 2000 bugs. The solution is to use the same proven software auditing and quality assurance method that leading software firms such as IBM, SAP, PeopleSoft, Oracle, Adobe, and others use to improve quality and reduce development costs and time—software inspections. Every error identified during an inspection means fewer errors to find during testing. This holds true when inspections are used in the Year 2000 software remediation lifecycle. Inspections can significantly increase the probability of finding Year 2000 noncompliances before testing and reduce the time and cost to complete Year 2000 projects. Properly executed, inspections provide a number of additional benefits:

- **Protect against application failures.** Inspections reduce the risk that an application will fail in production. When the Year 2000 arrives, an extraordinary number of date-related failures may occur simultaneously. A single error or failure can produce a ripple effect throughout operations. Minor errors can compound, resulting in damages far exceeding the sum of the individual errors. Unmanageable failures can paralyze operations. Each error caught during inspection, can mean far fewer problems later.
- **Reduce the impact of failures that may still occur and improve testing efforts.** Production errors are costly and time-consuming to fix. Inspections help focus testing efforts by identifying the likely failure points and the particularly tricky or risky areas in application logic, so contingency plans can be developed to handle potential errors. These preparations decrease the time needed to recover from errors, reducing their overall impact and cost.
- **Provide legal protection.** Inspections increase the chances that applications and systems will work reliably, as designed. In litigation over software failures, inspection and testing practices demonstrate the level of care taken by a company to reduce the risk of failure. The combination of inspection and testing efforts becomes a persuasive argument in avoiding legal liability.
- **Provide audit coverage.** Inspections produce documentation that demonstrates to auditors, insurers, government regulators, customers, and partners that extra measures have been taken to ensure software is Year 2000 compliant. Retaining inspection documentation will provide a paper trail to satisfy these entities and provide a foundation for legal defenses.

4. How Inspections Lowers Testing Risks and Costs

In a traditional software inspection, a programmer manually reviews modified source code, line by line.¹ Today, an inspection is a tool-assisted analysis and quality assurance review performed on modified software prior to testing. In the Year 2000 context, the purpose of inspections is to audit, validate, and document the compliance of remediated source code and to ensure that compliance specifications (standards for achieving compliance) and quality standards are observed. By providing the means to evaluate source code before testing, *the inspection process does not disrupt project schedules or ongoing processes. Quite the opposite: with inspections, overall cycle time is reduced, and resources can be redeployed.* In fact, with Reasoning's highly automated inspection toolset and process, inspections of millions of lines of code can now be completed in just a few days (see Section 7).

As shown in Figure 2, inspections are not meant to replace testing. Rather, they serve as an adjunct to testing by reducing the probability of "show stopper" defects making it to production. It is also a good policy to re-inspect code periodically after it is put back into production. This practice helps detect Year 2000 defects that infect compliant applications, including defects introduced during ongoing maintenance, and corruption through links with noncompliant systems and data.

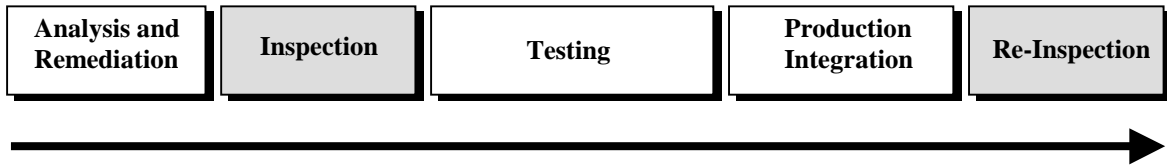


Figure 2: Where inspection fits in a Year 2000 project life cycle.

To understand how inspections save effort and associated costs, it is important to understand the testing process and the actions that occur when an error is discovered. Figure 3 depicts a typical testing phase with recurring efforts. Recurring tasks are performed each time a test is executed. It is far more costly to assemble an application for testing, set up a test environment, execute a test, detect an error, fix the error, and retest than it is to find the error in an inspection prior to testing. Multiple studies on software inspections have found that a company can achieve a return of 10 to 1 on its inspection investments by reducing the recurring testing efforts and the costs associated with finding and fixing defects during the testing process.² These findings are not surprising since an application often needs to undergo several testing cycles before it can be certified as Year 2000 compliant.

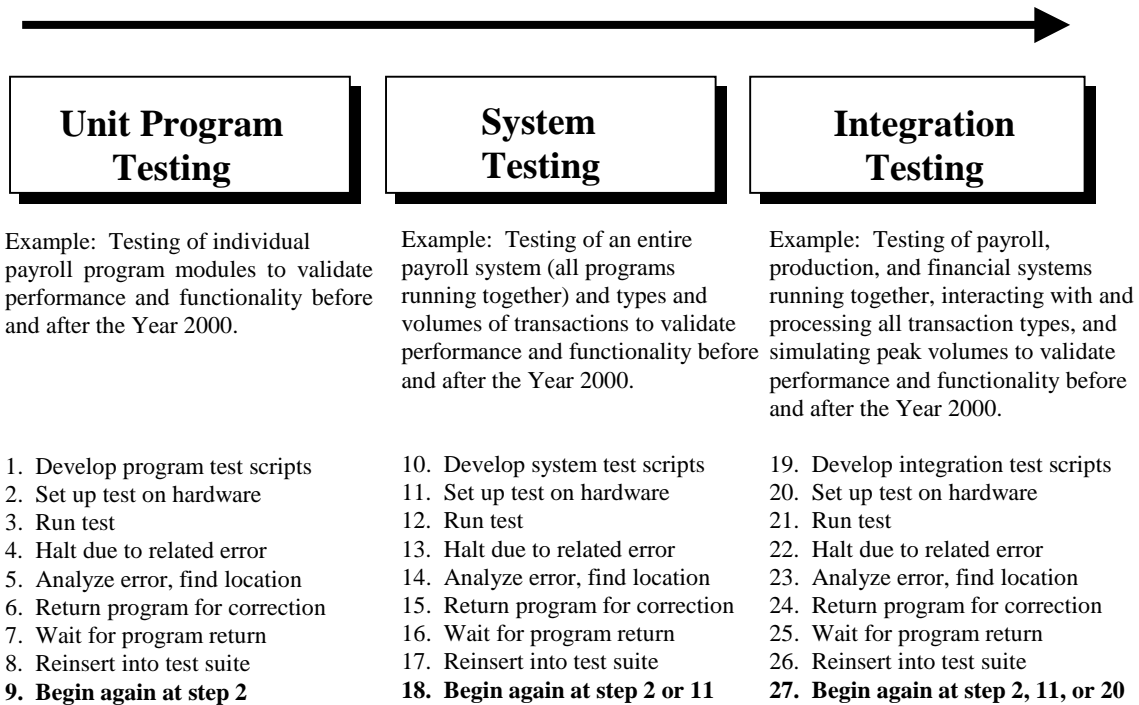


Figure 3: Year 2000 recurring testing efforts.

The greatest cost of an error that evades testing is likely to be the ramifications of the failure itself, which can range from inconvenience to significant damage to downstream operations to huge monetary losses. As the real-world examples cited in the Introduction of this paper illustrate, serious production errors can result in millions or billions of dollars in losses and liabilities. In the most likely scenario, an application will have already undergone several testing cycles before failing due to an unresolved production error.

Figure 4 demonstrates the effectiveness of software inspections as a means for improving the efficiency of a standard testing process. In this example, a very conservative percentage of changes still results in over 2,000 modifications in an application consisting of a million lines of code (LOC). A study of software reengineering projects found that programmers routinely make three errors for every 100 modifications, leading to 60 expected defects. Error types include omissions, modifying code that should not be changed, and making incorrect changes. Although overall productivity varied greatly, this study found little difference in the error rate between experienced and inexperienced programmers.

| | | Scenario 1 No Testing | Scenario 2 Testing Only | Scenario 3 Testing with Inspection |
|------------------------------------|------|--------------------------|----------------------------|--|
| Example Application | | | | |
| Application Size (LOC) | | 1,000,000 | 1,000,000 | 1,000,000 |
| % of Code Changed | 0.2% | | | |
| Number of Changes | | 2,000 | 2,000 | 2,000 |
| Expected Error Rate ³ | 3% | | | |
| Expected Defects | | 60 | 60 | 60 |
| Effect of Inspections | | | | |
| Inspection Efficiency ⁴ | 60% | | | |
| Errors Caught | | NA | NA | 36 |
| Effect of Testing | | | | |
| Testing Efficiency ⁵ | 30% | | | |
| Errors Caught | | NA | 18 | 7 ⁶ |
| Remaining Errors | | 60 | 42 | 17 |
| Probable Impact | | | | |
| Show Stoppers ⁷ | 5% | 3 | 2 | 1 |
| Mission Impact ⁷ | 20% | 12 | 8 | 3 |
| Minor ⁷ | 30% | 18 | 13 | 5 |
| Unresolved ⁷ | 45% | 27 | 19 | 8 |

Figure 4: The impact of Year 2000 software inspections.

The 60 percent inspection efficiency rate is taken from the published experience of the Hewlett-Packard Company, while the 30 percent testing efficiency rate assumes the use of production data as the basis for regression tests. The Probable Impact section categorizes the effect of errors that reach production. Show stoppers prevent the application from operating until they are resolved. Mission impact errors have a significant impact on application operations or cause damage to downstream operations. Minor errors result in inconvenience, while unresolved errors cause intermittent problems or remain dormant waiting to be triggered.

An examination of these three scenarios clearly demonstrates the benefits of using inspections to supplement testing efforts. At the depicted level of coverage, the standard testing procedures shown in scenario 2 are obviously inadequate. Overall testing efficiency would have to reach 72 percent to equal the combined benefits of inspection and testing as shown in scenario 3. Scenario 3 assumes that testing will detect an incremental set of errors in conjunction with inspection because some errors, such as faulty interactions with other software, cannot be found through inspection alone.

Although this data may seem to de-emphasize testing in favor of inspections, the combination of the two approaches provides the greatest benefits. By eliminating a large percentage of errors before testing, the inspection process reduces the number of errors that must be caught during testing (24 rather than 60), thereby reducing testing cycles and the number of errors that will slip into production.

5. How to Perform an Inspection

Performing an inspection is conceptually simple. In a Year 2000 project, an inspection is performed on a remediated application prior to testing. First, an extremely accurate and automated analysis tool must be used to rapidly analyze a remediated application to detect all date instances that are suspect. ***It is important to note that a different analysis toolset, process, and personnel should be used than the toolset, process, and personnel originally used to analyze and locate date defects in code.*** Otherwise, the value of the inspection is severely diluted and the fundamental benefit of an independent audit is forfeited.

Speed and accuracy are critical technical capabilities for Year 2000 inspections because one of the primary objectives is to reduce the time and cost of the Year 2000 project life cycle. According to the META Group, Reasoning's software analysis technology delivers the automation, speed, and accuracy needed to perform reliable inspections, and stands out as the yardstick by which other inspection offerings should be measured.⁸

The automated analysis step produces information or reports locating and describing suspect date occurrences along with problematic remediated code. All components of an application are analyzed together so that the consistency of remediation strategies used throughout the application can be verified. The information produced by the analysis step is then used to reconcile areas of concern. The inspector uses this information to review all date-sensitive items to determine whether they were modified or fixed correctly, and whether those fixes had any inadvertent side effects. As part of the review, the inspector also compares the remediated application to its initial compliance specification (compliance policy) to reconcile any discrepancies. For example, if the original compliance policy stated that all year fields should

be expanded to four digits but the remediation process failed to expand all the specified fields, this discrepancy would be identified and reconciled during the inspection. After noting all defects and discrepancies, the inspector can take several actions, depending on the particular scenario. For internally remediated applications, the application can be recycled for correction and further inspection. For vendor-packaged software or factory, or consultant-remediated code, the IT organization can demand corrective action or negotiate some other remedy. All inspection reports should be retained to establish a paper trail for future audits or litigation.

To streamline the inspection process, the inspection tool must not “over-report” or “under-report” suspected date instances. Over-reporting date instances increases workloads as valuable time is wasted examining and eliminating non-date suspects. Even worse, under-reporting compromises Year 2000 efforts by missing date instances that could cause production errors. An advanced tool like Reasoning/2000™ for Inspection can be configured to minimize over-reporting and automatically locate only those classes of Year 2000 faults specified by the compliance policy, thus increasing the speed and accuracy of the inspection. In addition it uses the latest technology, including advanced parsing, control and data flow analysis, and artificial intelligence to minimize under-reporting, and to provide accurate information to the user.

Who Should Perform an Inspection

A variety of people including programmers, systems analysts, consultants, and auditors can perform inspections, *but no one should ever inspect his or her own work*. To derive the full benefits of an inspection, reviewers must be objective, without any preconceived notions about an application or its remediation. The more familiar reviewers are with the remediated application, the less objective they can be and the more difficult it is to detect errors.

6. Inspection Scenarios

The scenarios below illustrate how IT organizations can use inspections to audit the Year 2000 compliance and quality of remediated software. Project managers may use inspections to review the work done by their own programmers, outside contractors, and software vendors. Auditors and legal staff may use inspections to monitor the performance of IT and Year 2000 progress, and to help create protective paper records in the event of future litigation.

Internally Remediated Applications

No IT organization, with the time and resources remaining, can create a Year 2000 test environment that is thorough enough to detect all errors introduced or overlooked during remediation. Given this limitation, there is a pressing need to detect the defects in a methodical manner rather than through hit-or-miss testing. Inspections can be used on internally remediated applications to:

- Verify compliance, ensuring that all affected dates have been found and fixed correctly without adversely impacting other functionality.
- Check for adherence to quality standards and compliance specifications.
- Direct testing efforts to likely failure points.

Offsite or Offshore Factory-Remediated Code

Offsite or offshore factory-remediated code is exposed to a variety of risks, including gaps in the vendor's quality assurance procedures (leading to poor code quality), lack of subject matter expertise (leading to incorrect remediation decisions), and language barriers or poor communications with internal staff (leading to incorrect remediation decisions). Inspections can be used to:

- Verify the compliance of remediated code and adherence to compliance specifications.
- Detect "error profiles" of common errors produced by the vendor's tools or techniques for early correction before errors proliferate.
- Monitor software quality. The vendor's manual versus automated methods, commercial versus those developed in-house, quality assurance processes, and standards enforcement all affect the quality and accuracy of the final product.

Consulting Project Audits

The remediated code produced by consulting vendors needs inspections because of various possible risks, including lack of subject matter expertise, inexperienced consultants, and gaps in the vendor's quality assurance procedures. The current IT resource shortage coupled with skyrocketing salaries has forced many consulting firms to staff with junior-level workers, especially when fixed-price contracts constrain salaries. Depending on the experience of the consultants, code quality can vary enormously.

Independent Validation and Verification (IV&V)

A growing number of commercial industry sectors such as financial services and public utilities, as well as government agencies are now required by regulators, legislation, or business policies to perform IV&V as a means of certifying Year 2000 compliance. Audits of software repairs via inspections provide organizations with a proven approach to satisfy IV&V requirements.

Vendor Software Packages

Most IT organizations rely on vendors to ensure the quality of their software, testing only internal modifications to the software. Consequently, these organizations are not equipped to test the functionality of a new release. Companies have already encountered problems with supposedly compliant software packages. For example, Household International, a financial services company, reported that it conducted tests on a reportedly compliant version of VisionPlus, a credit-card processing system from PaySys International Inc., only to find Year 2000 defects.⁹ Similar stories abound. Inspections can be used to:

- Verify the compliance of packages.
- Determine the level of testing required. If the inspection reveals compliance or quality problems, the organization should perform additional tests and use the inspection reports to pressure the vendor for fixes.

- Substitute for testing when testing is impractical or impossible. This scenario is likely to occur when a vendor delivers a release too late to permit testing or when IT does not have the test data or resources to perform testing.

Key Supplier Applications

Most companies depend on one or more key suppliers whose failure to provide goods or services could seriously impact corporate operations. For this reason, most companies are engaged in supplier management programs designed to ascertain supplier Year 2000 compliance and determine legal due diligence defenses. The problem is, without hard evidence, no company can blindly rely on a supplier's compliance claims. Confidentiality concerns can be addressed by having neutral third parties, such as consulting firms, perform the inspection. In some cases, contractual arrangements may give a company the right to audit its suppliers' operations. Inspections of key supplier applications are an excellent means of determining the supplier's true level of compliance. Inspections can be used to:

- Gain valuable insight into supplier Year 2000 operability.
- Verify the compliance of a key supplier's applications. This effort ensures that all critical applications have been properly fixed to avoid Year 2000 problems that could cause interruptions in services or supplies.
- Provide mutual benefits to a company and its suppliers. Although suppliers may initially resist inspection as too intrusive, the results can also benefit their business.

Maintaining Compliance After Remediation

Once an application has been remediated, tested, certified as compliant, and released to production, ongoing maintenance may cause it to become noncompliant in the future. Compliant modules may be inadvertently overlaid by noncompliant versions or be contaminated by noncompliant changes. Solid configuration management can help prevent module overlays, and training can help prevent programmers from making future non-compliant changes, but there is no way to guarantee that applications will remain compliant. For example, one large company accidentally discovered that 14 noncompliant changes had been introduced into certified, compliant code. Having to fix and retest contaminated applications not only wastes time and money, but also exposes weaknesses in internal controls and procedures, and provides the basis for lawsuits. Inspections can be used to monitor and audit the ongoing compliance of certified applications. While a random auditing program may suffice for most applications, critical applications should always undergo inspections to ensure their continuing compliance.

7. Inspection Case Studies

The following case studies summarize the results of five early inspection pilot projects performed by Reasoning. In four out of the five cases, the code had been previously remediated. In the remaining case the application was developed using Year 2000 compliant coding standards and was thought to be free of Year 2000 defects. The identities of the companies have been withheld at their request. All five companies are Fortune 500 class and have decided to perform Year 2000 software inspections on a broader set of applications.

Telecommunications Company #1

- Application: Various financial and accounting programs
- Lines of source code: 4,500,000
- Remediation method used: tool-assisted analysis and manual remediation
- Time to complete inspection: 3 weeks
- Year 2000 problems: 11 places in the code were found that would positively fail when processing dates that span the century; testing would not have detected these errors due to constraints on the test coverage plan. In addition this application was remediated using date field expansion to four digit years yet 1,637 date related variables with a 2-digit year were found as well as 114 hard-coded century “19s”.

Telecommunications Company #2

- Application: Service tracking system
- Lines of source code: 130,000
- Remediation method used: tool-assisted analysis and remediation
- Time to complete inspection: 4 days
- Year 2000 problems: This code had already been tested and put back into production. 21 places in the code were located that would positively fail when processing dates that span the century boundary. Description of failures:
 - One copybook that had a hard coded ‘19’ would have caused 9 programs to fail
 - Hard coded centuries were found which would have caused 2 programs to fail
 - Internal Sorts were found which would have caused 3 programs to fail
 - Date computations were found which would have caused 7 programs to fail
 - Date comparisons were found which would have caused 8 programs to fail

Utility Company

- Application: Service request and tracking system—this application was developed in accordance with Year 2000 compliant coding standards including four-digit DB2 Date Type year fields and was thought to be Year 2000 compliant
- Lines of source code: 2,334,011
- Time to complete inspection: 2 weeks
- Year 2000 problems: more than 1,000 fatal or serious errors were found including 254 date calculation errors, 192 date comparison errors, 537 erroneous move year errors, 105 date field errors, and 102 date windowing errors.

Aerospace Company

- Application: Purchase Order System
- Lines of source code: 106,249
- Remediation method used: tool-assisted analysis and manual remediation
- Time-to-complete inspection: 2.5 days
- Year 2000 problems: 6 fatal errors plus 12 potentially serious defects were found.

Health Insurance Company

- Application: Claims system
- Lines of source code: 30,065
- Remediation method used: tool-assisted analysis and remediation
- Time-to-complete inspection: 1 day
- Year 2000 problems: 51 fatal errors plus 20 potentially serious defects were found.

8. Conclusion

The value of inspections was best summarized by Watts Humphrey, a highly respected expert in software development. The following quote is from chapter 10 of his landmark book, Managing the Software Process (see footnote 1).

“Inspections are an important way to find errors. Not only are they more effective than testing in finding many types of problems, but they also find them early in the project when the cost of making corrections is far less.

Inspections should be a required part of every well-run software process, and they should be used for every software design, every program implementation, and every change made either during original development, in test, or in maintenance.”

The Year 2000 crisis poses risks to every business, no matter the size, location, or industry. There will not be enough time to test all mission critical software to the degree necessary to guarantee complete safety before the new millennium. Organizations need another safety net—software inspection. Inspections offer a powerful way to find errors, improve the quality and productivity of the software remediation process, and to help companies minimize the risks of Year 2000-related operational disruptions and their associated costs and legal liabilities.

9. Footnotes

1. The traditional inspection consisted of a peer review of a programmer's work to uncover problems and improve quality. Humphrey, Watts S. *Managing the Software Process*. Addison-Wesley, 1989, Chapter 10.
2. "...you can expect a yield of about 10 to 1 (not of total engineering costs, rather compared to what you would spend finding and fixing the same defects in test)." Grady, Robert B. *Successful Software Process Improvement*. Prentice Hall PTR, 1997, Chapter 13, p. 233.
3. In an unpublished study conducted by Ian S. Hayes of Clarity Consulting, Inc., South Hamilton, Massachusetts, programmers performing software reengineering work were found to commit an average of 3 mistakes for every 100 programming changes.
4. Inspections save 60 percent of rework costs. Grady, Robert B. *Successful Software Process Improvement*. Prentice Hall PTR, 1997, Chapter 13, p. 252.
5. Assumes 30% test path coverage, which is typical for test data taken from production.
6. $60 - 36 = 24 \times 30\% = 7$.
7. Jensen, Randall W., and Tonies, Charles C. *Software Engineering*. Prentice Hall, 1979, Chapter 5, p. 404.
8. "Y2K Inspection Services: Auditing Y2K Finders and Fixers." META Group, Inc. *Enterprise Data Center Strategies*, May 20, 1998.
9. Caldwell, Bruce "The Year 2000 Double Take—Project Managers Struggle to Cope as Software Vendors Fail to Deliver on Promises." *Information Week*, May 18, 1998, www.techweb.com.

For more information on the subject of software inspection, an excellent collection of references and information is available at <http://www.ics.hawaii.edu/~johnson/FTR/>. This site is maintained by Computer Science Professor Philip Johnson at the University of Hawaii.

10. About Reasoning, Inc.

Headquartered in Mountain View, California, Reasoning, Inc. is a leading provider of e-services that accelerate the readiness of applications for the Internet through a unique combination of automated and repeatable software inspection and transformation processes for large enterprises, independent software vendors and Internet companies. Reasoning's technology also ensures the reliability of mission-critical legacy applications. Founded in 1984, the company provides sales and services in North America and a growing network of subsidiaries and distributors in Europe and Asia-Pacific.

Reasoning, Inc.
700 East El Camino Real
Mountain View, CA 94040

Tel: +1 (650) 429-0350
Fax: +1 (650) 429-0222
Email: info@reasoning.com
Web: www.reasoning.com

©1999 Reasoning, Inc. All rights reserved. Reasoning, Reasoning5, Reasoning/2000 for Inspection, and the Reasoning logo are trademarks of Reasoning, Inc. All other product or service names may be trademarks of the companies with which they are associated.

Notes



Reasoning Inspector Plus

Software Quality and Reliability Services

Michael Jacobsen-Rey,
Brussels - November 3, 1999



Agenda

- Reasoning - the Company
- Why testing is not enough
- Software Inspection
- Inspector Plus
 - *Fundamentals*
 - *Inspection Classes*
 - *Service Methodology*
- Getting started

Company

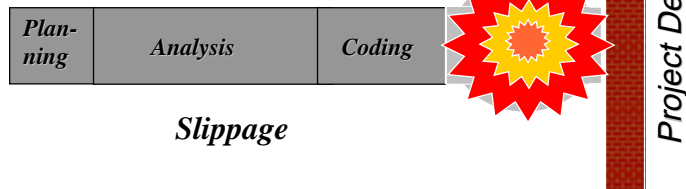
- Founded 1984, privately held
- Jim Treybig joins as Chairman of Board in 1996 (Founder & Former CEO of Tandem Computers)
- Headquarters in Mountain View, CA
- Offices in North America, Europe and Japan
- Stanford University re-engineering technology heritage
- Pioneer in automated software transformation and defect filtering technology
- Recognized as industry leader in 2nd-generation software analysis and transformation technology

Why testing is not enough

- Even the most rigorous testing performed by U.S. companies finds only 85% of all software defects. [*Capers Jones, 1997*]
- Most testing programs only test 30% to 40% of an application. [*The Standish Group, 1998*]
- “Testing is never finished, only abandoned” [*Encyclopedia of Software Engineering*]
- Testing is capital and labor intensive

Pressure to shortchange testing

- Comes at end of Software Development Life Cycle
- Project overruns lead to cuts in testing to meet deadlines



Software inspection

Software inspection is an egoless, peer review of software to find problems and improve quality.

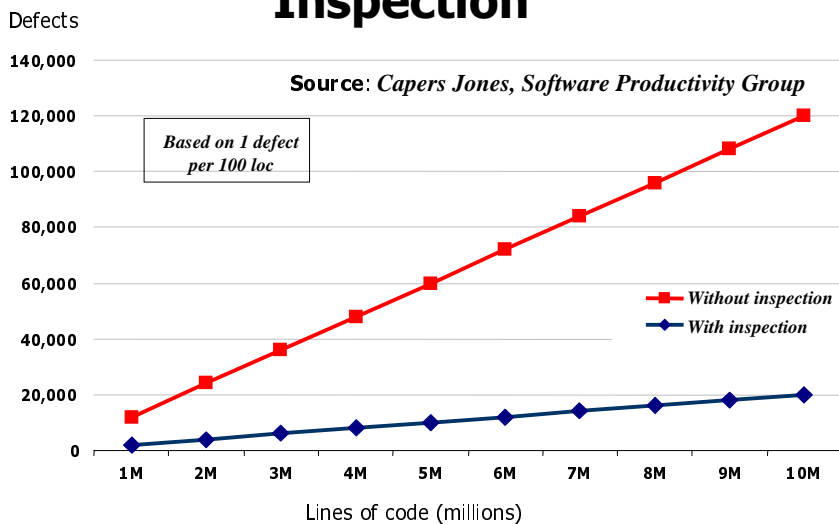
Proof reading.....

Software inspections can be more effective than test

- NASA Goddard Space Flight Center
- Code inspection was found to be more effective than either functional testing or structural testing

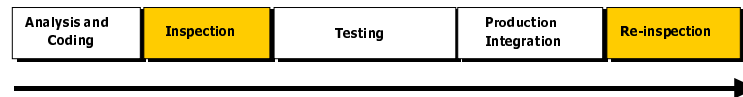
V.R. Basili and R.W. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering

Less defects after Inspection



Inspection & QA Process

- Ideally occur after development/maintenance is complete and prior to testing
- May occur after testing
- May occur during maintenance



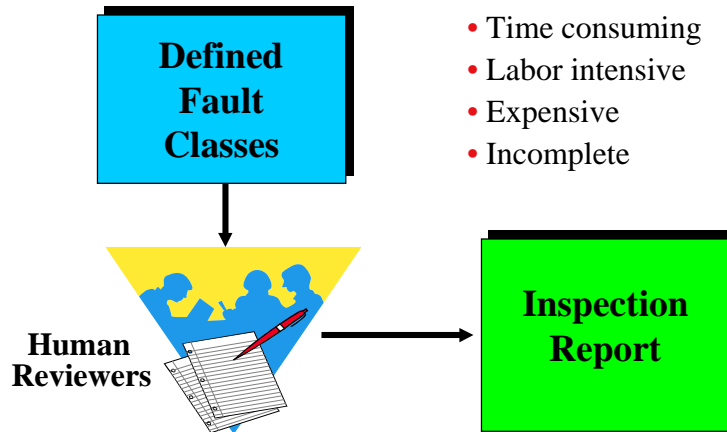
Inspection accelerates development and reduces costs

- Inspection finds up to 65% of all defects prior to testing, thus reducing test time and cost.¹
- The cost of fixing a defect in testing is 10 times the cost of finding a defect prior to testing.²
- Inspections save 60% of rework costs.³

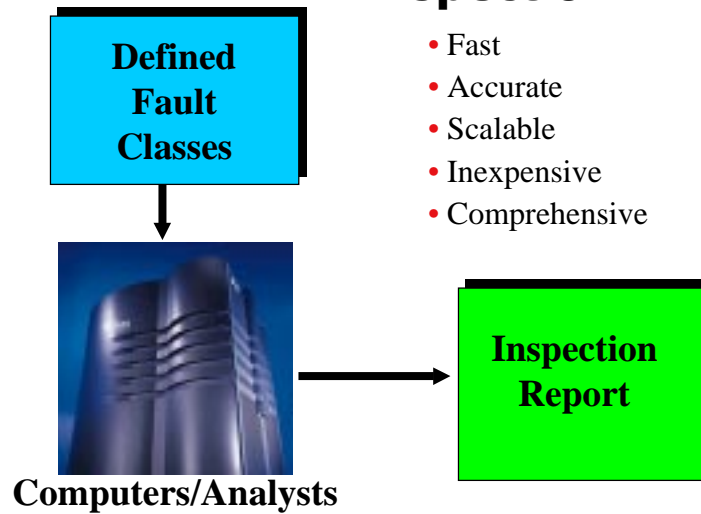
1. Capers Jones, Software Productivity Research, Inc.

2, 3. Robert B. Grady, *Successful Software Process Improvement*, Prentice Hall PTR, 1997.

Manual Software Inspection



Automated Software Inspection



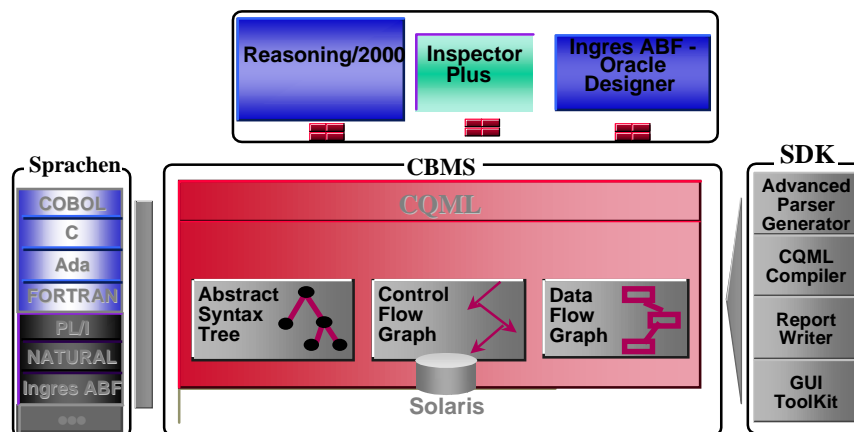
Inspector Plus

Inspector Plus
Technology and
Services



- Recognizes Defects
- Improves Reliability
- Measures Software Fragility
- Checks on Coding standards
- Provides Process Improvement
- Ongoing Process possible
- Fast and Accurate

Fundamentals



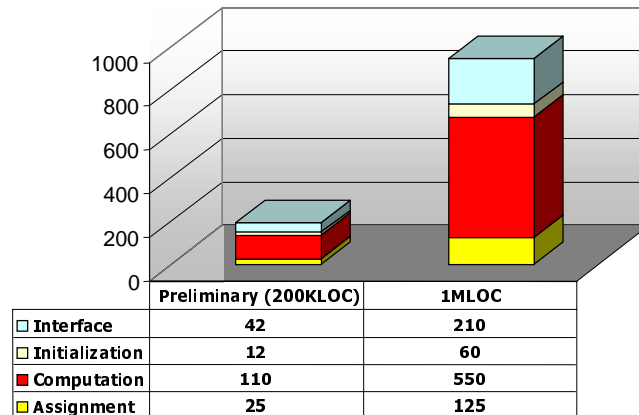
Defect inspection classes (samples)

- Interface (causes application to crash)
 - Incorrect number/type/size of arguments
 - SQL/CICS Return code unchecked
- Initialization (leads to unpredictable behavior)
 - Uninitialized data element
 - Record used fields partially set
- Computation (causes incorrect behavior)
 - Arithmetic overflow
 - Unsigned subtraction
 - Alphanumeric to numeric

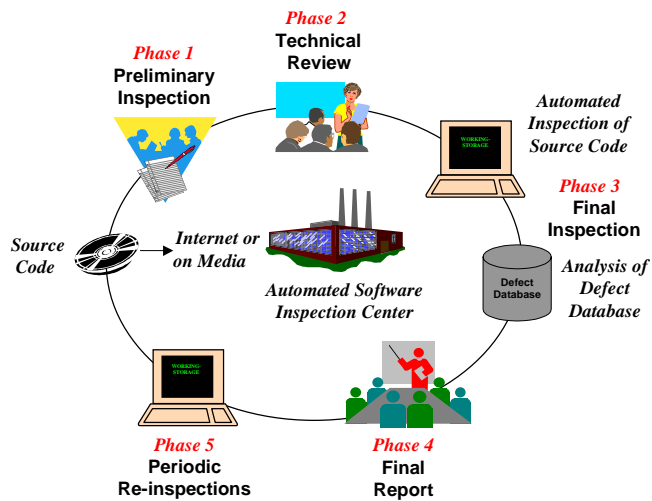
Defect inspection classes (cont.)

- Assignment (results in data corruption)
 - Size truncation
 - Precision truncation
 - Sign truncation
- Fragility (makes program error-prone)
 - GO TO out of perform range
 - GO TO into perform range
 - GO TO out of paragraph
 - Deeply nested IF statement
 - Unreachable paragraph (dead code)

Potential violations per MLOC



Inspection process



Deliverables

- Comprehensive Reporting
 - Software Defect Reports
 - Software Fragility Report
 - Defect Classification Report (Defect DB)
 - Standards Compliance Report
- Action Plan
 - Software Defect Action Items
 - Software Fragility Action Items
 - Software Process Action Items
- Final Report and Recommendations to Management

Automated inspection case study

- 300K lines of code
- 5 company employees inspecting the same code in parallel, manually

Results:

- Reasoning's service was 1/3 the cost
- 1/30 the time, and
- Found all errors (manual effort missed 3)

Benefits of Inspector Plus Services

- Independent review
- Leading Inspection technology
- Proven Inspection methodology and expertise
- Minimum impact on development and project management resources
- High productivity, fast turnaround
- High quality, low cost

Summary

Inspector Plus:

- Improves software reliability
- Offers insurance against costly business disruptions
- Decreases the costs of software development and maintenance
- Speeds the time to deliver new releases



Getting started



- Preliminary inspection, including consultancy
- Fixed price engagement
- Includes value added management consulting:
 - Defect database analysis
 - Software development process analysis and recommendations

The final word

“Inspections are an important way to find errors. Not only are they more effective than testing in finding many types of problems, but they also find them early in the project when the cost of making corrections is far less.

Inspections should be a required part of every well-run software process....”

Watts Humphrey, Managing the Software Process, Addison-Wesley, 1989, Chapter 10

Test Environment for Terminal Certification

Dr. Cristian Radu
Integri n.v.,
Leuvensesteenweg 325,
Zaventem, BELGIUM
e-mail : cradu@integri.be

1

Outline of the talk

- Motivation
- Terminal environment
- Test management
- TERTIO - Terminal test environment tool
- Graphical User Interface
- Implementation aspects of the test environment

QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 2

Motivation

The processing of the terminal is described in its *functional specifications*, which typically cover:

- the interface with each entity, specifying the format of the command/response pairs or the format of the messages exchanged with the host computers;
- the data structures managed by the terminal, like the Transactions Journal, Terminal Parameters, and Red Lists;
- the protocol of each transaction supported by the terminal;
- the messages displayed for the Merchant and the Cardholder;
- the sequence of keys that is needed to trigger a certain function provided by the terminal;
- the initialization and error recovery procedures.

The behaviour of the terminal is completely determined through its interfaces with the entities composing the terminal environment.

Terminal certification makes sure that the implementation of the terminal complies with its functional specifications ⇔ Need for certification tools:

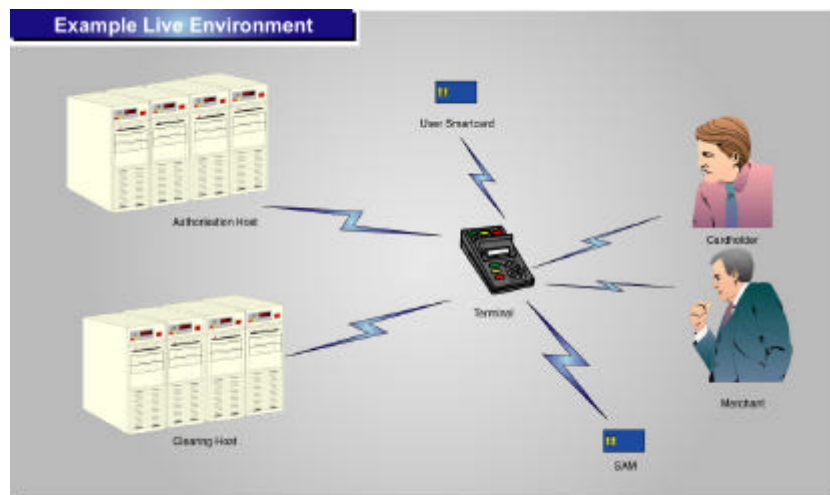
TERTIO = TERminal Test EnvIrOnment tool

QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 3

Terminal environment



QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 4

It can be noticed that a number of interfaces can be defined between the terminal and each party that interacts with the terminal:

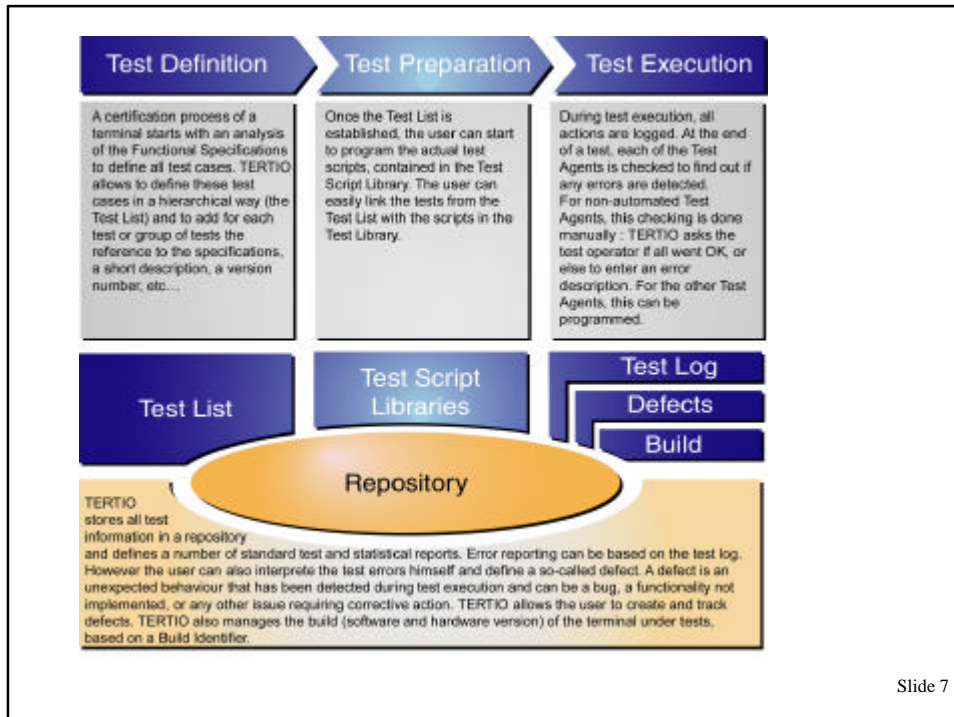
- *User card interface* -- it consists of the set of all the command/response pairs exchanged between the terminal and the card.
- *SAM interface* -- it is the set of all the command/response pairs exchanged between the terminal and the SAM.
- *Man-machine interface* -- it consists of the specialised display, keyboard, and printer that allow the interaction of the POS terminal with the Cardholder and/or the Merchant.
- *Authorisation Host interface* -- it represents the set of all messages exchanged between the Authorisation Host and the terminal for authorising debit and credit transactions.
- *Clearing Host interface* -- it represents the set of all messages exchanged between the Clearing Host and the terminal during a collection transaction.

The behaviour of the terminal is completely determined through its interfaces with the entities composing the terminal environment.

Test management

The certification process starts with an analysis of the functional specifications of the terminal to define all the *representative* test cases. Representative tests should fulfil the following requirements:

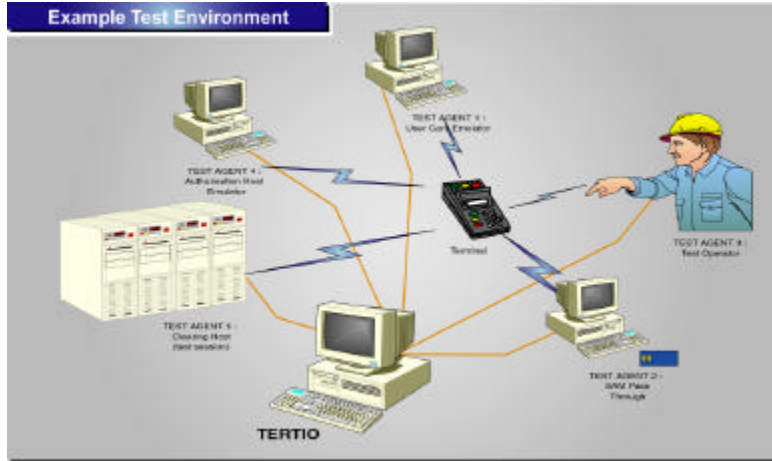
- *effective*: the main purpose is that tests should find errors;
- *exemplary*: the tests should be representative for as many cases as possible;
- *maintainable*: the effort to adapt tests, if the functional specifications of the terminal change should be minimal;
- *economical*: the tests should be written and executed with as little effort as possible, such that the time needed to execute tests should be minimal.
- *robust*: the tests should not be sensitive to the state of the unit under test. If the state does not correspond to the pre-requisite of the test, the test is able to recover from this situation and bring the unit into this desired state.
- *repeatable*: it must be possible to repeat the test. Note however that this does not imply that exactly the same input/output will occur because this is not always possible, e.g., due to random numbers generated by the unit under test, or due to counters maintained by the unit under test. However it is important that the same test case can be repeated.



TERTIO - Terminal test environment tool

The *test environment* is defined like a set of Test Agents interacting with the terminal to be tested under the control of a specialised program referred to as the *Terminal Test Environment Tool*, or shortly TERTIO. A *Test Agent* in the test environment plays the role of an entity in the terminal environment. A Test Agent is featured by the following four characteristics:

- A Test Agent emulates the external behaviour of the corresponding entity from the terminal environment by the point of view of the interface with the terminal. Thus, the terminal under test does not sense the difference between interacting with the Test Agent or with the corresponding entity.
- If the data coming from the terminal is erroneous, the Test Agent will detect the anomaly.
- A Test Agent is able to induce errors in the data forwarded to the terminal. Examples of errors are format errors in the response of the user card, a time-out in operating a sequence of keys expected in a certain status of the terminal, or a wrong MAC verification value returned by the SAM.
- A Test Agent can be controlled by TERTIO during all of the three stages of execution of a test.



QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 9

TEST SCRIPTS

```

***** PRE_REQUISITE
BufferStr1 = TestProp("Test Node Path")
Call XT_Set(THALES_ITMC_STREAM, THALES_ITMC_TIMEOUT, "UC.Emulation.CurrentTest",
BufferStr1)
Call XT_Set(THALES_ITMC_STREAM, THALES_ITMC_TIMEOUT, "CSM.PassThrough.CurrentTest",
BufferStr1)
Call XT_Set(HOST_ITMC_STREAM, HOST_ITMC_TIMEOUT, "AuthorisationHost.Data.CurrentTest",
BufferStr1)

***** EVENT SCRIPT
RespAuthorisation
    Call T_Set ("Response.msg_crc", "824A")
    Call Compile_Msg

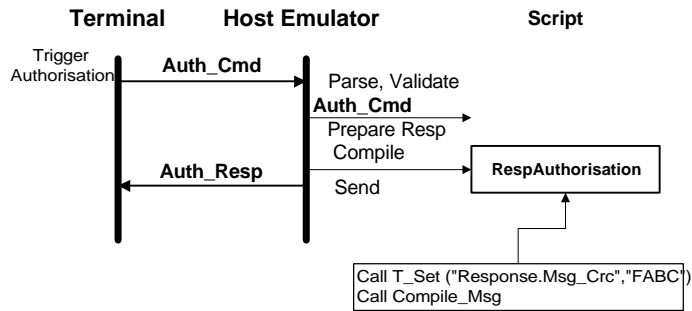
***** POST_REQUISITE
Action = "Please check if the terminal has rejected the Auth_Resp"
Call TestGrid_Add("<Screen.ERROR>")
Call TestGrid_Show
    
```

QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 10

Triggering of an event-script on a remote Test Agent



QWE '99, November 3, 1999, Brussels

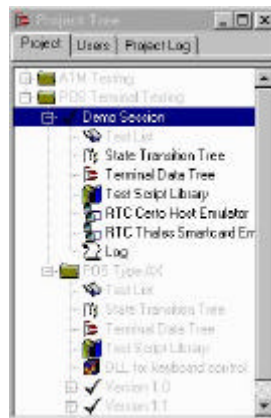
©Integri, 1999

Slide 11

Graphical User Interface

Project Tree

It defines various test sessions. Each test session contains the references to the objects that are loaded when a test session is opened, e.g., terminal data tree, state transition tree, test list, test library, log, and links to the Test Agents.



QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 12

Terminal Data Tree

It stores terminal parameters, red lists, screen messages to be displayed by the terminal, and records of the Transactions Journal. This information can be used for the programming of the test scripts and of the State Transition Tree of the terminal.

| Name | IFM | Value |
|---------------------|---------|------------------|
| Parameters | | |
| Terminal_Identifier | 8h | 0000013A |
| Software_Version | 4n | 0102 |
| Hardware_Version | 2n | 0100 |
| Host_Address | | 194.78.77.201 |
| Transient | | |
| CurrentCard | | |
| PIN | n(2-20) | 6703000000000101 |
| Expiry_Date | YYMM | 9912 |
| Service_Code | n(3) | 100 |
| Discretionary_Data | n(1-20) | 00001 |
| Amount | n6 | 000155 |

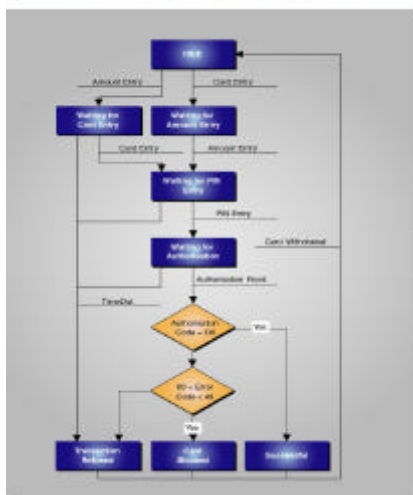
6703000000000101

QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 13

Flow Chart Diagram



QWE '99, November 3, 1999, Brussels

Representation In Tertio

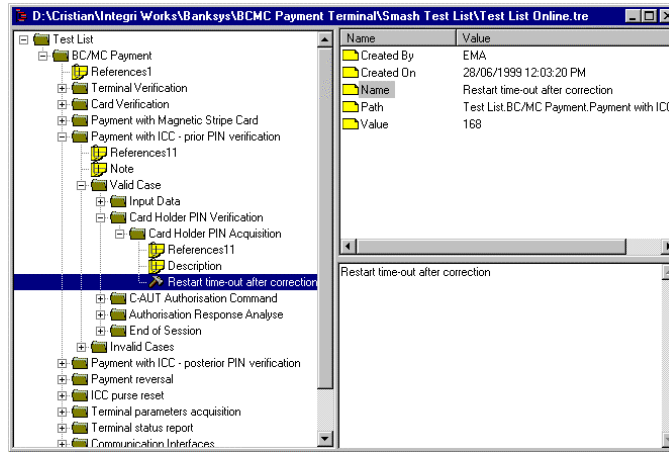
The script in the State Transition Tree (STT) contains the necessary code to control the test environment. E.g. the code executed on the PIN_Entry event commands the appropriate Test Agent to enter a PIN. Making use of the State Transition Tree (STT) facilitates the writing of test scripts, because the script only needs to indicate in which state the terminal has to be set before the action, specific to the given testis executed.

©Integri, 1999

Slide 14

Test List Tree

It describes the tests that are necessary to be executed on the terminal under test in order to certify it according to its functional specifications.



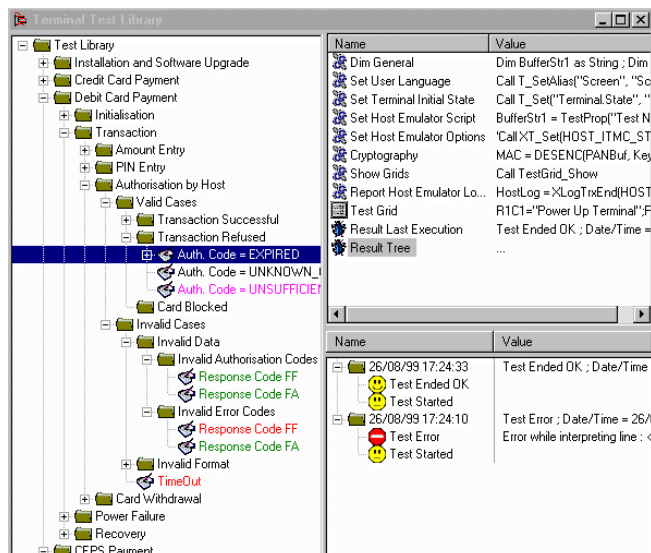
QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 15

Test Script Library

It contains the actual test scripts to be executed. TERTIO is provided with an extended set of software libraries offering cryptographic, numeric, bit-wise, and string operations, including support for manipulating data trees.



QWE '99, November 3, 1999, Brussels

©Integri, 1999

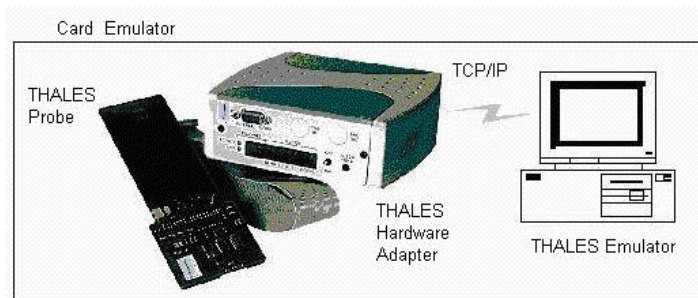
Slide 16

Test Grid Modal Window

In case a Test Agent cannot be automated, test grids are displayed for the Test Operator in order to assist him managing the Test Agent.

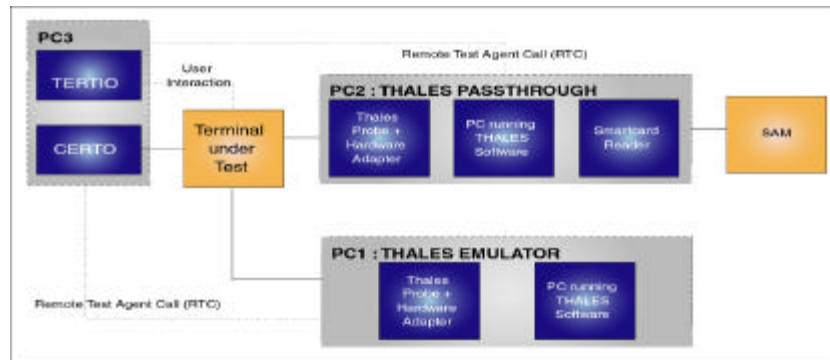
| Action | Display | Comments |
|---|--|---|
| Power Up Terminal | READY | |
| Wait for max. 2 Sec | merchant : PLEASE ENTER AMOUNT : ____ cardholder : PLEASE WAIT | |
| Enter Test Card 0001 | merchant : PLEASE ENTER AMOUNT : ____ cardholder : PLEASE WAIT | |
| Enter Amount 1.55 and press OK on merchant keyboard | merchant : PLEASE ENTER AMOUNT : 0001.55 cardholder : 0001.55 EURO - ENTER PIN : ____ | |
| Enter PIN 1234 on cardholder keyboard | merchant : PLEASE ENTER AMOUNT : 0001.55 cardholder : 0001.55 EURO - ENTER PIN : XXXX | Check that PIN cannot ready (asterix or other symbol) |
| DO NOT Press OK | merchant : PLEASE ENTER AMOUNT : 0001.55 cardholder : 0001.55 EURO - ENTER PIN : XXXX | |
| Wait for maximum 10 seconds | BEEP BEEP BEEP merchant : TRANSACTION REFUSED - NO PIN | |

Implementation aspects of the test environment



THALES software package

- Definitions Tree data structure
- Card Tree data structure
- DLL that implements the behaviour of the card
- PC running THALES
- THALES Hardware Adapter -- STAR 1150



QWE '99, November 3, 1999, Brussels

©Integri, 1999

Slide 19

Integration with other INTEGRI tools

- THALES Test Tool: to prepare physical test smart cards;
- THALES Emulator: to fully emulate a smart card;
- THALES Pass-Through: forwards requests from the terminal to a physical smart card, optionally modifying response data from the smart card before returning it to the terminal;
- CERTO -- to emulate a host or a cash register interface.

Hardware and software requirements

- Windows NT @ 4 SP 3
- 64 Mbyte RAM
- 40 Mbyte harddisk space
- TCP/IP if RTC is used

Slide 20

Presentation

Title : TestFrame, organising the use of record and playback tools with Action Words.

Category : Advanced Testing Techniques

Abstract

CMG has introduced an entirely new method to the market that improves the use of record and playback tools in an end user environment: Automated testing with action words. This method is five years old now and is used by hundreds of people in countless mainly large and complex projects. Using available test tools standard test software has been developed which improves the use of those tools. In separate files (i.e. spread sheets) called clusters, records are defined containing an action word and all the parameters used during this action. The action word is linked to one or more functions in tailor made test scripts. The main advantages of this method are:

- Early start of test development: since the test logic is defined in separate clusters the definition of test cases can start as soon as the development of the system has started.
- Test software is easy to maintain: Test script are programmed, not record. This implies a clear structure of the technical test products
- Separation between test scripts and test logic: Test cases can be accessed by people who are not experts in test tools or even testing in general
- Clear functional report back of the results of the test in separate test reports

TestFrame

Testing with Action Words

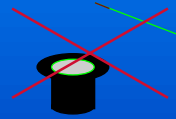
Chris Schotanus
CMG
THE NETHERLANDS
e-mail: Kris.Schotanus@cmg.nl

TestFrame

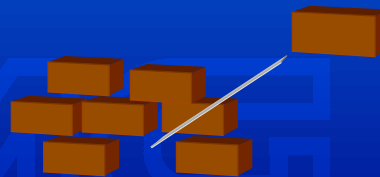
CMG's integrated method for:
test development
test automation
test organization

The Action Word Method

- not a magic wand

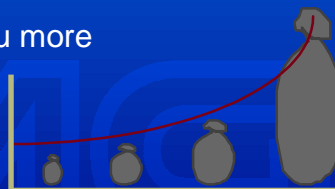


- just a brick in the wall

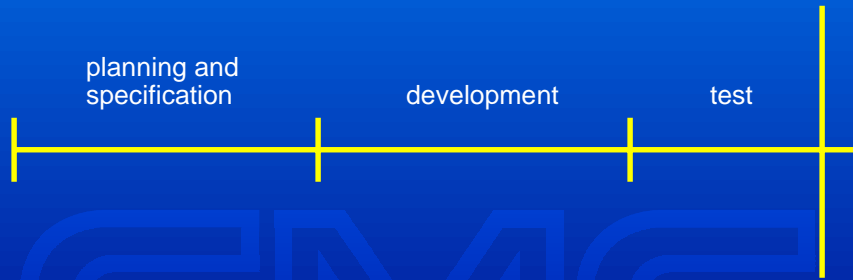


testing is necessary

- there are always faults
- faults are risks until they are found
- finding them later will cost you more

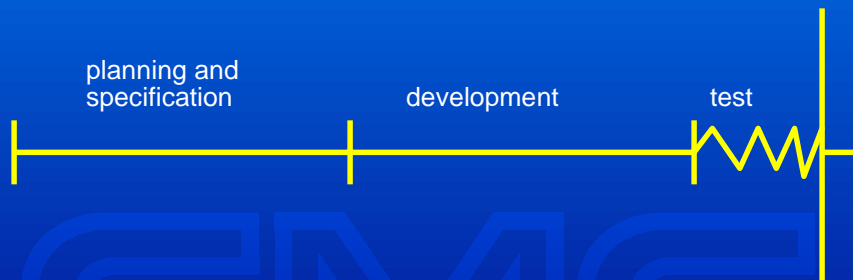


but testing often gets under pressure



CMG

but testing often gets under pressure



CMG

quality aspects software (source ISO/IEC 9126)

- functionality ← test
- reliability ← test
- usability ← assess
- efficiency ← test
- maintainability ← assess
- portability ← partly test

CMG

common experienced problems with testing and test automation

- costly
- time consuming
- boring to do
- you can't start early
- often neglected
- difficult to manage:
 - what is the progress
 - what is the quality
- the proper resources (users, specialists) are not available when needed
- automated scripts hard to maintain
- ...

CMG

some reasons for automating the test process

- saving time by repeating tests automatically
- less boring work
- consistent test execution
- higher reliability of outcomes

GNMG

record and playback

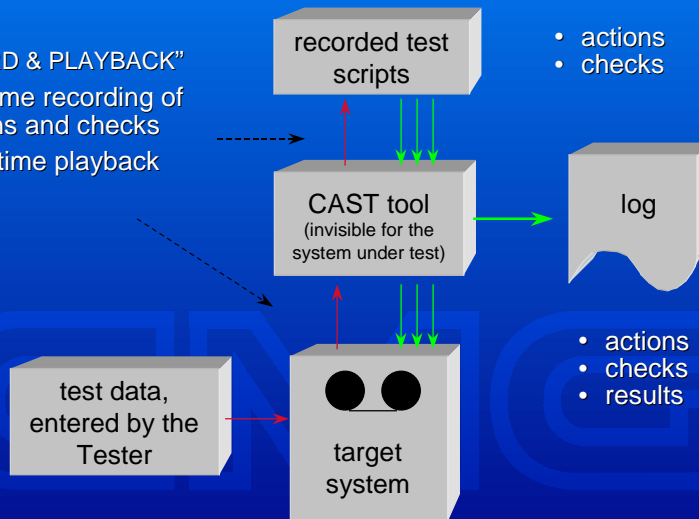


(not our approach...)

schematic overview record & playback

“RECORD & PLAYBACK”

- one time recording of actions and checks
- multi time playback



But beware of pitfalls...

- highly sensitive for maintenance
- test cases difficult to access
- dependant on a working system
- only suitable for on-line systems

an alternative: Action Words



re-usable test products in TestFrame

| A | B | C | D |
|----------------------|---------|-------|-----|
| ... | | | |
| <i>transfer</i> | Houston | Klein | 210 |
| <i>check balance</i> | Klein | 210 | |
| <i>transfer</i> | Savy | Klein | 150 |
| <i>check balance</i> | Klein | 360 | |
| ... | | | |

functional

navigation script
(test tool)

cluster with scenarios
(in a spreadsheet)

technical

```
case action
  "transfer": ...
  "check balance": ...
end
```

separation of test development and test execution

- test development aimed at the production of “clusters”

- input and expected results
- test language with “action words”
- in spreadsheets

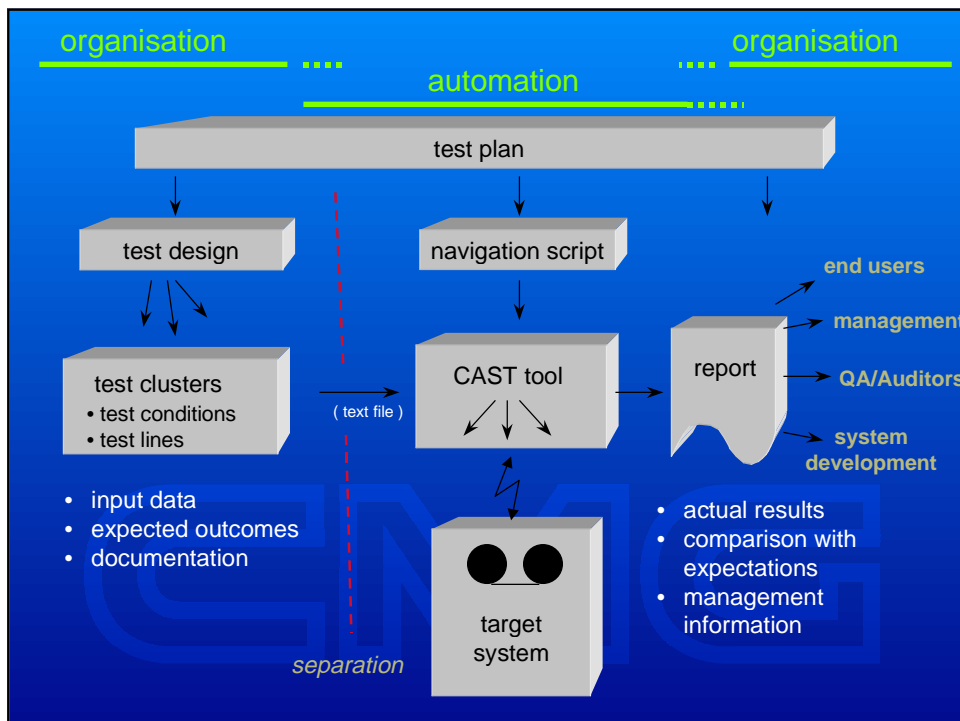
| A | B | C | D |
|----------------------|----------------|--------------|--------------|
| ... | | | |
| transfer | Houston | Black | \$210 |
| check balance | Black | \$210 | |
| ... | | | |

- automatic execution by a “navigation script”

- written in the script language of the cast tool
- general part: the engine
- specific part: the action words

functional
technical

case action of
“transfer”: ...
“check balance”: ...
end case



example of a cluster

action words test data

| | | | | |
|----------------------|---|------------|------------|---------|
| <i>cluster</i> | EXAMPLE OF A TEST CLUSTER | | | |
| <i>version</i> | 1.0 | | | |
| <i>author</i> | Hans Buwalda | | | |
| <i>section</i> | 1. Entering clients and balances | | | |
| | last name | first name | account nr | balance |
| <i>enter client</i> | Green | John | 458473948 | 1500 |
| <i>enter client</i> | Wood | Anna | 422087596 | 2100 |
| <i>section</i> | 2. Money Transfers | | | |
| | from | to | sum | |
| <i>transfer</i> | 458473948 | 422087596 | 500 | |
| <i>transfer</i> | 422087596 | 785793025 | 1201 | |
| <i>section</i> | 3. Checking names and numbers | | | |
| | account nr | last name | first name | |
| <i>check name</i> | 458473948 | Green | John | |
| <i>check name</i> | 422087596 | Wood | Anna | |
| | account nr | sum | | |
| <i>check balance</i> | 458473948 | 1000 | | |
| <i>check balance</i> | 422087596 | 1399 | | |

documentary

input

expected output

example of a cluster level report (1)

```

=====
cluster name      :    EXAMPLE OF A CLUSTER
cluster version   :    1.0
cluster author    :    Hans Buwalda

script name       :    CMG CAST Navigation Script for MS Test
script version    :    1.0
script release date :    February 1997

run date and time :    3-03-97 13:39:16
=====
SECTION 1 - Relation management
-----
1 ( 6 ): enter client      Johnson Jean   458473948    f 1500
2 ( 7 ): enter client      Juet   Christian 422087596    f 2100
3 ( 8 ): enter client      Savy   Anne     785793025    f 1700
4 ( 9 ): autonum          on
5 ( 10 ): enter client     Puk    Pierre   &keep 01    f 1000
  
```

example of a cluster level report (2)

```

11 (20): check name      76392763      Puk   Pierre
12 (23): check balance  458473948    f 1000
13 (24): check balance  422087596    f 1400
->FAILED:                f 1399

=====
end of cluster          :      EXAMPLE OF A CLUSTER
finished at             :      3-03-97 13:39:26
time used               :      15

number of cluster lines :      26
number of test lines   :      13
number of checks       :      10
number of checks       :      10
number passed          :      9
number failed          :      1
percentage passed      :      90 %

failed at test lines (see above report):
13
=====

```

example test condition

```

nr      description
...
3.11   it is checked that the exit date is after the entry date
...

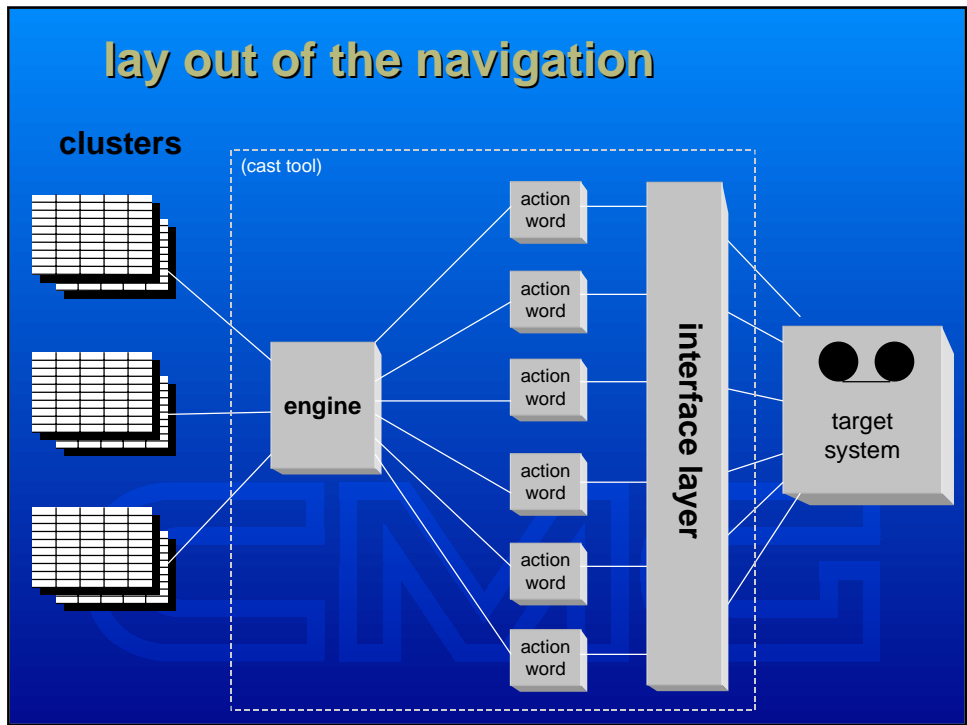
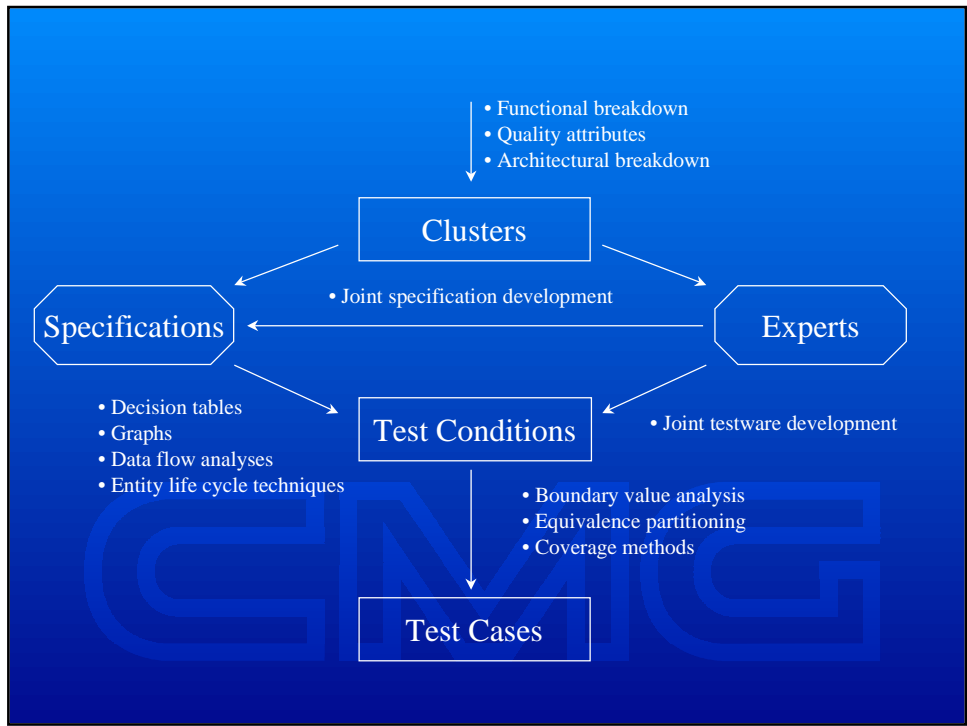
```

coupled with the actual test lines in the cluster

```

begin condition      3.11
enter employment     name      entry date      exit date
check error message  Bill Goodfellow  1996-10-02     1996-10-01
                    The exit date must be after the entry date.

```



the method does not rely on a specific tool

- Winrunner/XRunner
- QA Run
- Hiperstation
- MS/Visual Test
- Rational
- ATF
- Autotester
- SilkTest
- ...

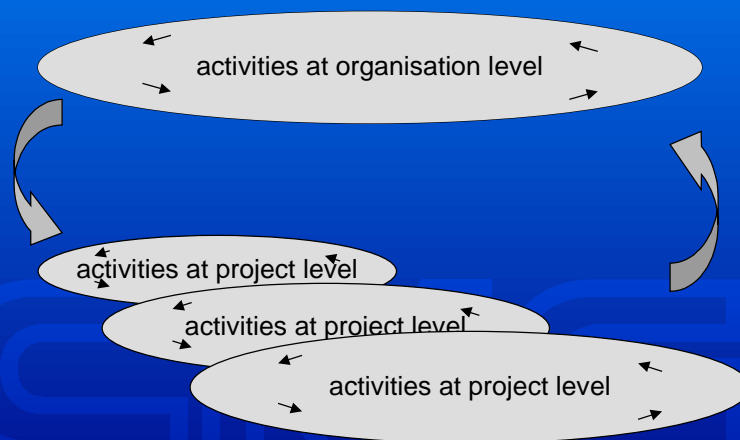
Strategic Context of TestFrame



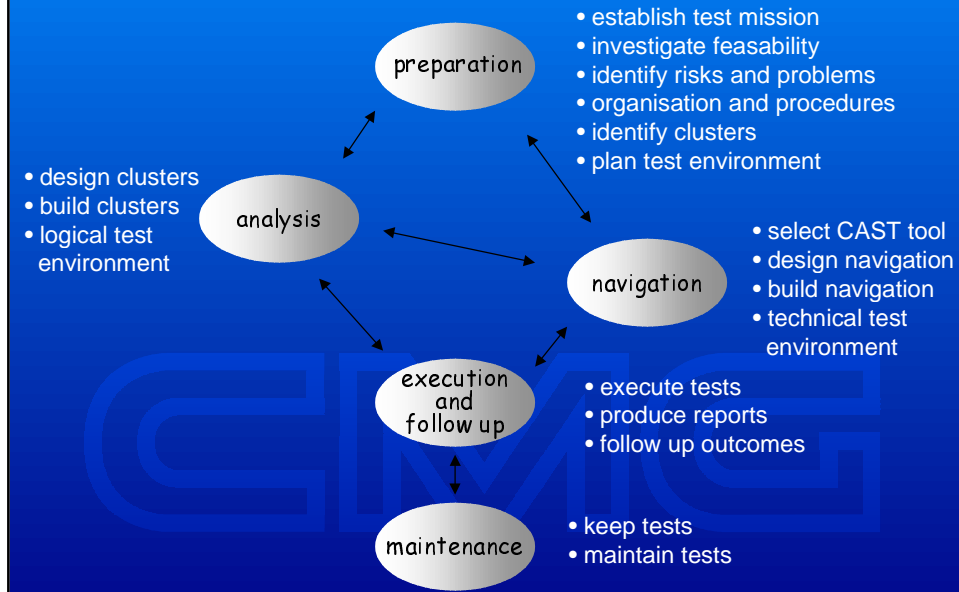
tasks related to the method (examples)

- within a project
 - test consultancy: formulating the test mission (functional, technical)
 - test management: managing the change process
 - test analysis: production of the test clusters
 - navigation: production of the scripts for automatic execution
- general, at the organisation level
 - support on the method
 - keeping navigation script
 - keeping test clusters

Organisation and project level



Activities at project level



Activities at Organisation Level

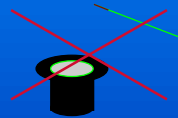
- one or more pilots
- training and handbooks
- resourcing (pooling, hiring)
- auditing and reviewing
- r&d
- development of common products
- ...

some special applications

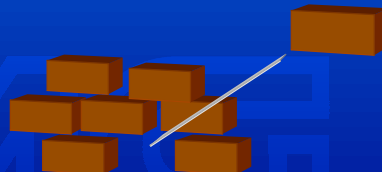
- performance and multi user testing
- testing of batch systems
- testing of large scale conversion like year 2000 and euro
- regression testing
- test generation
- test result analysis
- fault tracking
- interface testing


The Action Word Method

- not a magic wand




- just a brick in the wall






Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 1

Test Automation



Levering your test results



ps_testWare
Software Testing Service

Test Automation

Levering your test results

Geert Pinxten
Development Co-ordinator




Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 2

toc Intro Start Phase II se III. TB Less. Info

Agenda

- **Introduction to test automation**
- **How it started**
- **Phase II: data-driven**
- **Phase III: script programming**
- **Automated script generation**
- **Introduction to ps_testware**

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 3

ps_testWare
Software Testing Service

toc

Intro

Start

Phase II

Phase III

TB

Less.

Info

Introduction to test automation

Scope

Where does it start?

Why automating tests?

Benefits

Problems

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 4



toc

Intro

Start

Phase II

Phase III

TB

Less.

Info

ps_testWare
Software Testing Service

Scope

- GUI-applications
- Test Automation with record-playback tools
- System test -> Functional test

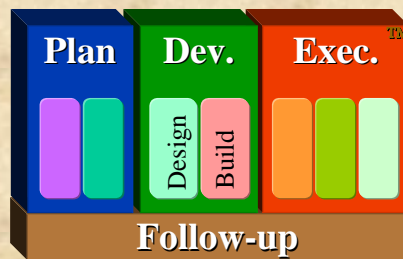


Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 5

ps_testware
Software Testing Service

toc Intro Start Phase II Phase III TB Less. Info

Where does it start?



- **Design**
 - Translate requirement into test
 - Logical
 - Physical
 - Test script
- **Build**
 - Record test procedure

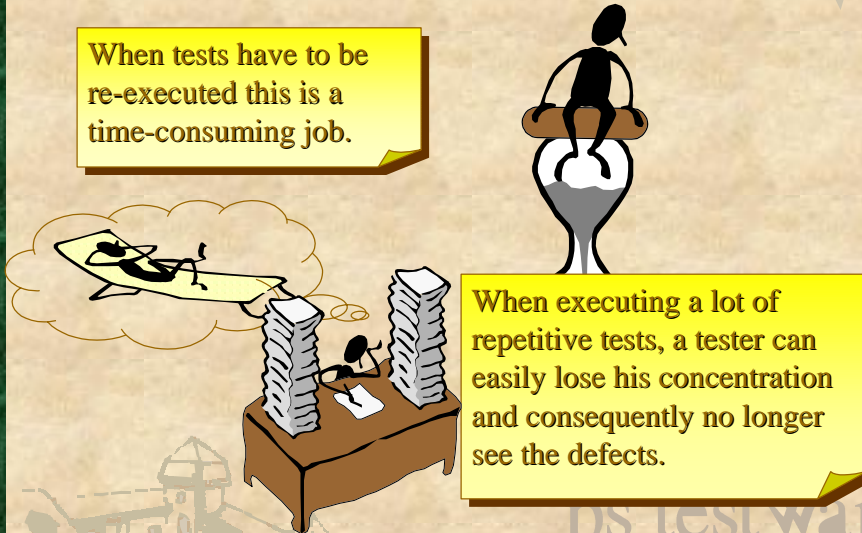
Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 6

ps_testware
Software Testing Service

toc Intro Start Phase II Phase III TB Less. Info

Why automating tests ?

When tests have to be re-executed this is a time-consuming job.

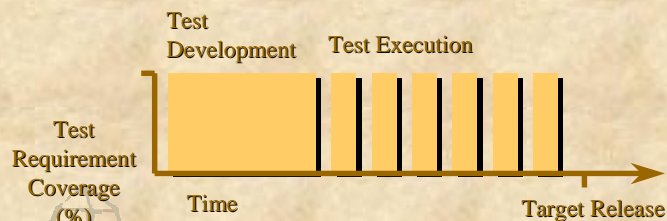


When executing a lot of repetitive tests, a tester can easily lose his concentration and consequently no longer see the defects.

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 7

Benefits

- Automated testing allows focussing on dynamic areas in the application under test
- Automated testing speeds up the turnaround time



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 8

Problems

- **Automated tests are not cheap**
- **Automated tests postpone the finding of errors**
- **Automated tests reveal only regression errors**
- **Automated tests requires maintenance**



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 9

ps_testWare
Software Testing Service

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

ps_testWare
Software Testing Service

How it started

Case description
Approach for scripting
Conclusions



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 10



toc
Intro
Start Phase II
Phase III
TB
Less.
Info

Case description

- **Home banking application R1.0**
 - Different releases planned
 - 50.000 home users
 - Critical success factors
 - Stability
 - Performance
 - Functional correct
- **RAD development**

Approach for scripting

- **Way of working**
 - Started automating on first release
 - Planned intermediate releases (#5)
 - Test designs
 - Basic programming techniques in scripting

AUT

Test Script

Defect solving

Conclusions

- **Problems**

- Unexpected functional changes
- Initial software quality low
- Many intermediate releases
- Low anticipation on what will be received

- **Resulting**

- Test maintenance 284% of budgeted time (15% of lap time)
- Regression test took too long
- Test designs not up-to-date

Limited success

Phase II: Data-driven

- Case description
- Approach for scripting
- Conclusions



Case description

- **Home banking application R2.0**
 - Functionality extended
- **Improvements in test process**
 - Introduction of intake criteria
 - Communication rules
 - Test designs changed
 - Still started from first release



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 15

ps_testware
Software Testing Service

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

Approach for scripting

- **Data-driven**



- **Scripting standards**

- Understandable scripts (comments, indentations)
- Anticipate on application changes
- Test scripts should test
- ...

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 16

ps_testware
Software Testing Service

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

Conclusions

- **Improvement**
 - Test script maintenance: 130%
 - Full regression test fastest
- **Problems**
 - Many intermediate releases (#20)
 - Low anticipation on way defects solved
 - Test Designs still not up-to-date
- **New problem**
 - Testers require programming skills (IO-files)

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 17

ps_testWare
Software Testing Service

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

Phase III: Script programming

Approach for scripting Conclusions

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 18

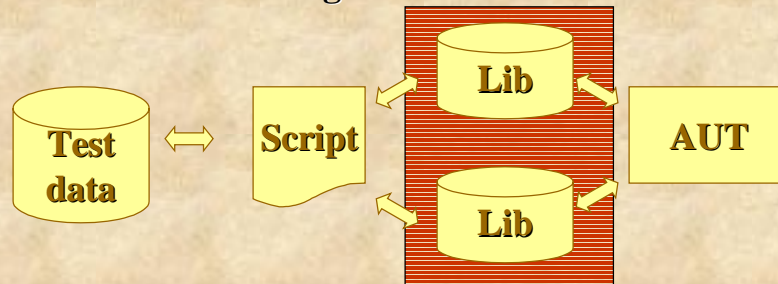


toc
Intro
Start Phase II
Phase III
TB
Less.
Info

Approach for scripting

- **Way of working**

- Start automating on first release!



- AUT separated from test script

- Technical script design
- Use of function libraries

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 19

ps_testware
Software Testing Service

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

Conclusions

- **Improvement**

- Test script maintenance: 110%
- Full regression test in two man-days
- Re-usability

- **Disadvantage**

- Contact tester/AUT lower

- **Main problem**

- Advanced programming techniques required
- Test designs still not updated

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 20

ps_testware
Software Testing Service

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

ps_testWare
Software Testing Service

Automated script generation

TOP BOX

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 21

toc Intro Start Phase II se III. TB Less. Info

What is TOP BOX

```

graph TD
    TD[Test design] --> AF[Action file]
    TD --> IF[input file]
    AF -.-> IF
    AF --> TB[TOPBOX]
    IF --> TB
    TB <--> AUT[AUT]
  
```

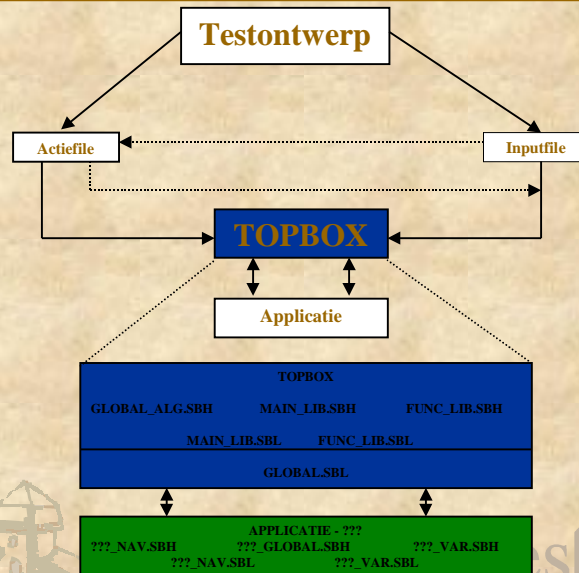
- Automatic test procedure generation
- Translation of design into script
- Same principles as in phase III

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 22

toc Intro Start Phase II ase III TB Less. Info

Top Box/Libraries



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 23

Advantages

- **Standardisation of all scripting activities**
- **Maintenance limited**
- **Test designs always up-to-date**
- **Testers do not need test tool knowledge**
- **Independent of test tool**

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 24

ps_testWare
Software Testing Service

Lessons learned



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 25

1994 - 1999
5th Anniversary
Software Testing Service

toc Intro Start Phase II se III. TB Less. Info

Lessons learned

- **Test Automation is software development**
- **Staffing requirements**
 - Testers test
 - Developers automate
- **Automation takes time**
- **Script maintenance effort can be controlled**

ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 26


toc Intro Start Phase II ase III TB Less. Info

[toc](#)
[Intro](#)
[Start](#)
[Phase II](#)
[Phase III](#)
[TB](#)
[Less.](#)
[Info](#)

ps_testWare



Software Testing Service

Questions




Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 27

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 28

ps_testWare
Software Testing Services





ps_testware
Software Testing Services

Tiensesteenweg 329
 B-3010 Leuven
 Tel.: +32 (16) 35.93.80
 Fax: +32 (16) 35.93.88
 e-mail: ps_testware@compuserve.com
<http://www.pstestware.com>

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 29

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

ps_testware

- 6** • **Started in 1991**

5 • **1993: Tools & technical services**

 - Tool Training
 - Coaching

6 • **1995: Methodological Services**

 - Consultancy

15 • **1996: Software Testing Services**

 - Test Assignments
 - Test Plan
 - Test Report
- 25** • **1997: Software Testing Services Suite**

 - Test Assessments
 - Y2K training

33 • **1998: PSTI**

 - Office @NL
 - Total Outsourcing
 - Partnerships

+ 28? • **1999: Testing Hill™ Campus**

 - ...

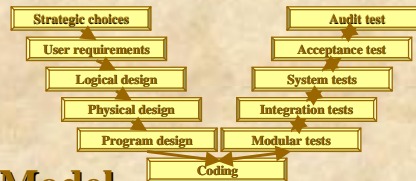
Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 30

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

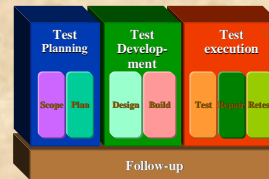
Our Business

- **Structured Software Testing**

- **Methodology**



- **Implementation Model**



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 31

Our Services




- **Training (see ps_testware institute)**
- **Coaching**
- **Consultancy**
- **Outsourcing (now also Total Outsourcing)**

Provided by:

- **Test Engineers**
- **Test Consultants**
- **Management Consultants**

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 32

Our Products

- **Test Assessment**
- **Test Assignment**
- **Test Plan**
- **Test Report**
- **Test Advice**
- **Test Audit**
- **Test Pack™** 
- **Test Laboratory** 
- **Tools** 

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 33

ps_testware
Software Testing Service

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

References

- **Kredietbank**
- **Barco Graphics**
- **Exact Maatwerk**
- **ING Bank**
- **Bank Card Company**
- **Janssen
Pharmaceutica**
- **Tessa**
- **Europese Raad**
- **Lernout & Hauspie**
- **Origin**
- **Specs**
- **Gemeentekrediet**
- **Siemens**
- **ING 2**
- **Yokogawa**
- **Link**
- **Alcatel Bell**
- **Mobistar**

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 34

ps_testware
Software Testing Service

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

Credo

ps_testware's first responsibility goes to the customers who use our services. Our services must be of high quality and must be a reference for our customers. In line with our primary business, Structured Software Testing, we may not indulge in pressure, quantity or quick profit.

We are responsible to our members, the men and women who work with us. Every member must be respected as an individual and must be rewarded personal and fair. We must support our members via a competent management, an adequate working environment and proper working conditions. Our members must have the means to provide and receive feedback, allow them and the organisation to learn continuously. We must support our members in their family responsibilities. Our actions must be just and ethical.

Our final responsibility is to our stockholders. Our business must make a sound profit. We must innovate and continuously improve our methods and techniques. We must develop new services and implement them effective and efficient. We must create reserves to provide for adverse times. Our stockholders must receive a fair return on their investments.

The Mission

To offer the best solution to quality problems of computer systems by using its test expert knowledge in a professional way.

Best solution: the solution that provides the highest contribution.

Test expert knowledge: the intellectual asset of ps_testware, a profound and complete knowledge regarding verification and validation (testing).

Professional: the courage to really provide what has been promised.

ps_testware

Software Testing Services

Tiensesteenweg 329
B-3010 Leuven
Tel.: +32 (16) 35.93.80
Fax: +32 (16) 35.93.88
e-mail: ps_testware@compuserve.com
<http://www.pstestware.com>

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 37

toc

Intro

Start

Phase II


Phase III

TB


Less.

Info


Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 1



Test Automation




Levering your test results





ps_testWare
Software Testing Service

Test Automation

Levering your test results



Geert Pinxten
Development Co-ordinator

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 2

toc Intro Start phase II se III. TB Less. Info

Agenda

- **Introduction to test automation**
- **How it started**
- **Phase II: data-driven**
- **Phase III: script programming**
- **Automated script generation**
- **Introduction to ps_testware**



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 3

ps_testWare
Software Testing Service

| |
|-----------|
| toc |
| Intro |
| Start |
| Phase II |
| Phase III |
| TB |
| Less. |
| Info |

ps_testWare

Software Testing Service

Introduction to test automation

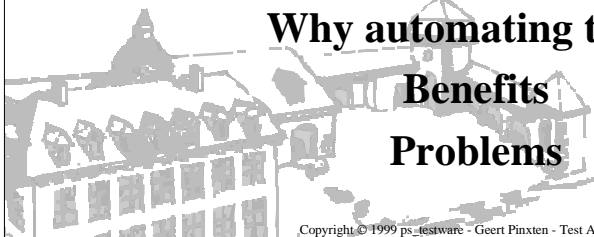
Scope

Where does it start?

Why automating tests?

Benefits

Problems



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 4



| |
|-----------|
| toc |
| Intro |
| Start |
| Phase II |
| Phase III |
| TB |
| Less. |
| Info |

Scope

- **GUI-applications**
- **Test Automation with record-playback tools**
- **System test -> Functional test**

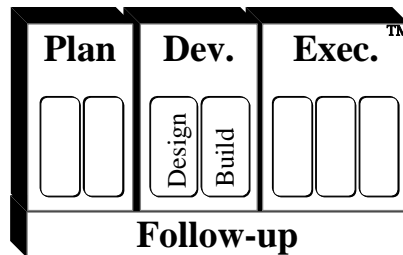


ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 5

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Where does it start?



- **Design**
 - Translate requirement into test
 - Logical
 - Physical
 - Test script
- **Build**
 - Record test procedure



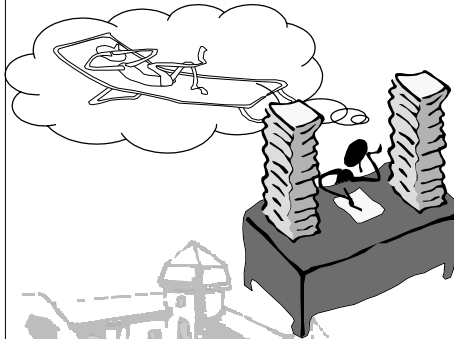
ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 6

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Why automating tests ?

When tests have to be re-executed this is a time-consuming job.



When executing a lot of repetitive tests, a tester can easily lose his concentration and consequently no longer see the defects.

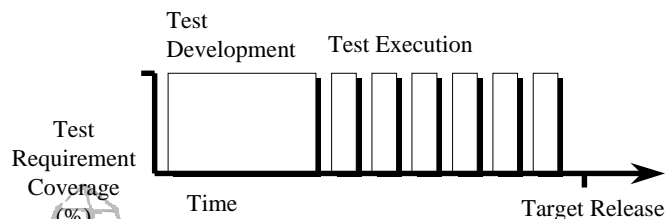
Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 7

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Benefits

- **Automated testing allows focussing on dynamic areas in the application under test**
- **Automated testing speeds up the turnaround time**



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 8

ps_testware
Software Testing Service

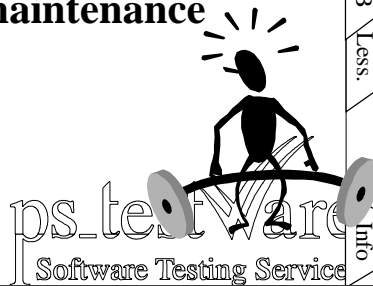
toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Problems

- **Automated tests are not cheap**
- **Automated tests postpone the finding of errors**
- **Automated tests reveal only regression errors**
- **Automated tests requires maintenance**



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 9



toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

ps_testWare

Software Testing Service

How it started

- Case description**
- Approach for scripting**
- Conclusions**



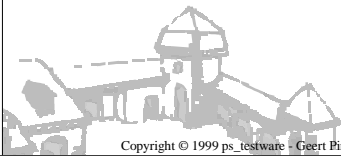
Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 10



toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Case description

- **Home banking application R1.0**
 - Different releases planned
 - **50.000** home users
 - **Critical success factors**
 - Stability
 - Performance
 - Functional correct
- **RAD development**



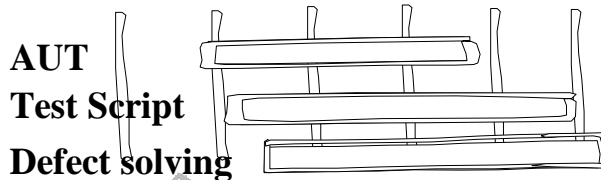
Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 11

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Approach for scripting

- **Way of working**
 - Started automating on first release
 - Planned intermediate releases (#5)
 - Test designs
 - Basic programming techniques in scripting



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 12

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Conclusions

- **Problems**

- Unexpected functional changes
- Initial software quality low
- Many intermediate releases
- Low anticipation on what will be received

- **Resulting**

- Test maintenance 284% of budgeted time (15% of lap time)
- Regression test took too long
- Test designs not up-to-date

Limited success



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 13

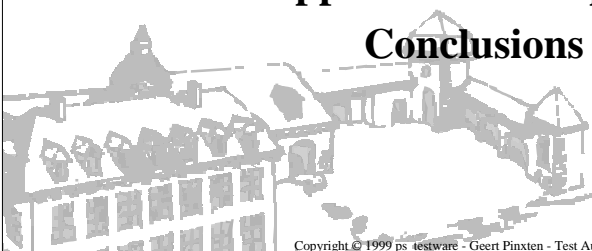
ps_testWare
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

ps_testWare
Software Testing Service

Phase II: Data-driven

Case description
Approach for scripting
Conclusions



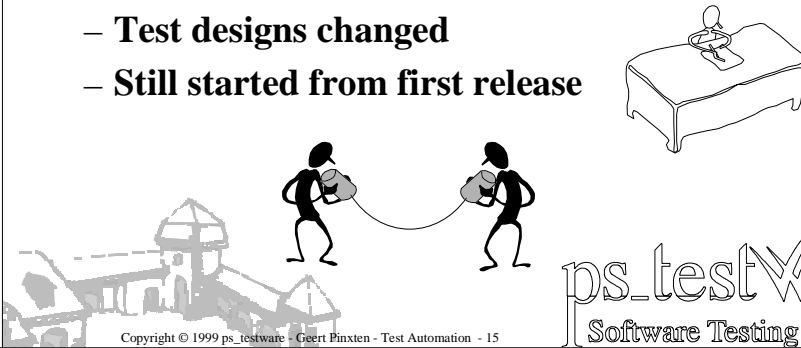
Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 14



toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Case description

- **Home banking application R2.0**
 - Functionality extended
- **Improvements in test process**
 - Introduction of intake criteria
 - Communication rules
 - Test designs changed
 - Still started from first release



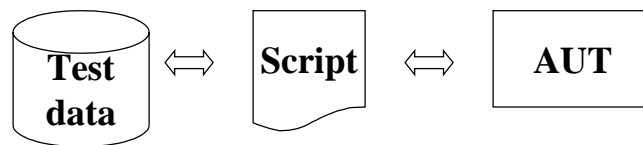
Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 15

ps_testWare
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

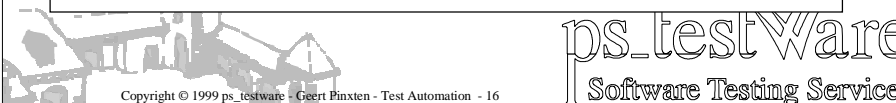
Approach for scripting

- **Data-driven**



- **Scripting standards**

- Understandable scripts (comments, indentations)
- Anticipate on application changes
- Test scripts should test
- ...



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 16

ps_testWare
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Conclusions

- **Improvement**
 - Test script maintenance: **130%**
 - Full regression test fastest
- **Problems**
 - Many intermediate releases (#20)
 - Low anticipation on way defects solved
 - Test Designs still not up-to-date
- **New problem**
 - Testers require programming skills (IO-files)



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 17

ps_testWare
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

ps_testWare

Software Testing Service

Phase III: Script programming

Approach for scripting

Conclusions



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 18

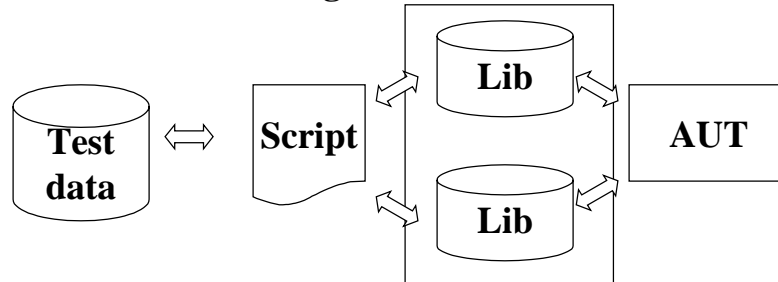


toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Approach for scripting

- **Way of working**

- **Start automating on first release!**



- **AUT separated from test script**

- **Technical script design**
- **Use of function libraries**

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 19

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Conclusions

- **Improvement**

- **Test script maintenance: 110%**
- **Full regression test in two man-days**
- **Re-usability**

- **Disadvantage**

- **Contact tester/AUT lower**

- **Main problem**

- **Advanced programming techniques required**
- **Test designs still not updated**

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 20

ps_testware
Software Testing Service



toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

ps_testWare

Software Testing Service

Automated script generation

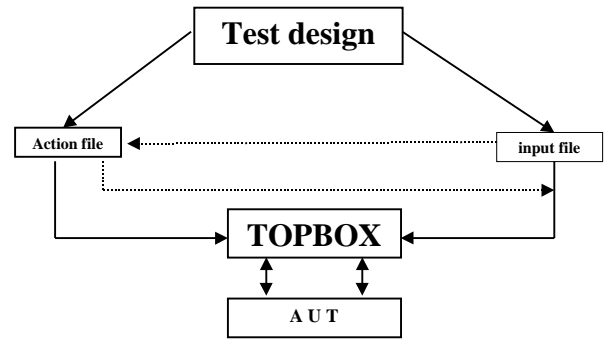
TOP BOX

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 21

toc Intro Start phase II phase III TB Less. Info



What is TOP BOX



```

    graph TD
      TD[Test design] --> AF[Action file]
      TD --> IF[input file]
      AF <-.-> IF
      AF --> TB[TOPBOX]
      IF --> TB
      TB <--> AUT[AUT]
  
```

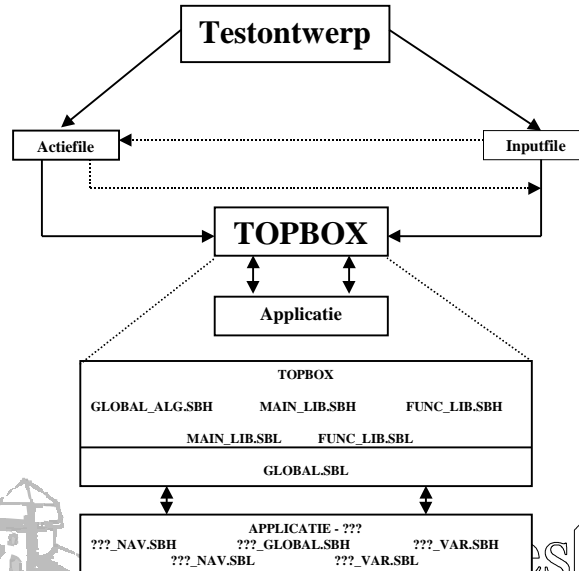
- Automatic test procedure generation
- Translation of design into script
- Same principles as in phase III

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 22

toc Intro Start phase II phase III TB Less. Info

Top Box/Libraries



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 23

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Advantages

- **Standardisation of all scripting activities**
- **Maintenance limited**
- **Test designs always up-to-date**
- **Testers do not need test tool knowledge**
- **Independent of test tool**

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 24

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

ps_testWare
Software Testing Service

Lessons learned




Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 25

toc Intro Start phase II se III. TB Less. Info

Lessons learned

- **Test Automation is software development**
- **Staffing requirements**
 - Testers test
 - Developers automate
- **Automation takes time**
- **Script maintenance effort can be controlled**



ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 26

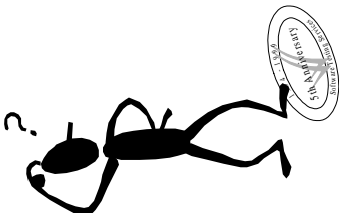

toc Intro Start phase II se III. TB Less. Info

| | | | | | | | |
|-----|-------|-------|----------|------------|----|-------|------|
| toc | Intro | Start | Phase II | Phase III. | TB | Less. | Info |
|-----|-------|-------|----------|------------|----|-------|------|

ps.testWare

Software Testing Service

Questions

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation 27

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 28







ps_testWare
Software Testing Services

Tiensesteenweg 329
B-3010 Leuven
Tel.: +32 (16) 35.93.80
Fax: +32 (16) 35.93.88
e-mail: ps_testware@compuserve.com
<http://www.pstestware.com>

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 29

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

ps_testware

- 6** • **Started in 1991**

5 • **1993: Tools & technical services**

 - Tool Training
 - Coaching

6 • **1995: Methodological Services**

 - Consultancy

15 • **1996: Software Testing Services**

 - Test Assignments
 - Test Plan
 - Test Report
- 25** • **1997: Software Testing Services Suite**

 - Test Assessments
 - Y2K training

• **1998: PSTI**

 - Office @NL
 - Total Outsourcing
 - Partnerships

• **1999: Testing Hill™ Campus**

+ 28? - ...

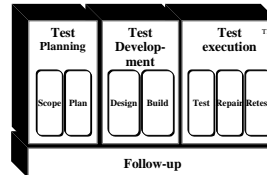
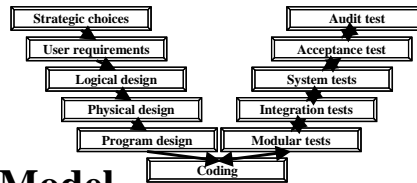
ps_testWare
Software Testing Service

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 30

toc
Intro
Start Phase II
Phase III
TB
Less.
Info

Our Business

- **Structured Software Testing**
- **Methodology**
- **Implementation Model**



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 31

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Our Services

- **Training (see ps_testware institute)**
- **Coaching**
- **Consultancy**
- **Outsourcing (now also Total Outsourcing)**

Provided by:




- **Test Engineers**
- **Test Consultants**
- **Management Consultants**

Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 32

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Our Products

- **Test Assessment**
- **Test Assignment**
- **Test Plan**
- **Test Report**
- **Test Advice**
- **Test Audit**
- **Test Pack™** 
- **Test Laboratory** 
- **Tools** 



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 33

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

References

- **Kredietbank**
- **Barco Graphics**
- **Exact Maatwerk**
- **ING Bank**
- **Bank Card Company**
- **Janssen
Pharmaceutica**
- **Tessa**
- **Europese Raad**
- **Lernout & Hauspie**
- **Origin**
- **Specs**
- **Gemeentekrediet**
- **Siemens**
- **ING 2**
- **Yokogawa**
- **Link**
- **Alcatel Bell**
- **Mobistar**



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 34

ps_testware
Software Testing Service

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info

Credo

ps_testware's first responsibility goes to the customers who use our services. Our services must be of high quality and must be a reference for our customers. In line with our primary business, Structured Software Testing, we may not indulge in pressure, quantity or quick profit.

We are responsible to our members, the men and women who work with us. Every member must be respected as an individual and must be rewarded personal and fair. We must support our members via a competent management, an adequate working environment and proper working conditions. Our members must have the means to provide and receive feedback, allow them and the organisation to learn continuously. We must support our members in their family responsibilities. Our actions must be just and ethical.

Our final responsibility is to our stockholders. Our business must make a sound profit. We must innovate and continuously improve our methods and techniques. We must develop new services and implement them effective and efficient. We must create reserves to provide for adverse times. Our stockholders must receive a fair return on their investments.

The Mission

To offer the best solution to quality problems of computer systems by using its test expert knowledge in a professional way.

Best solution: the solution that provides the highest contribution.

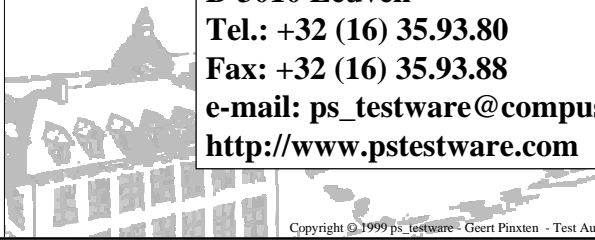
Test expert knowledge: the intellectual asset of ps_testware, a profound and complete knowledge regarding verification and validation (testing).

Professional: the courage to really provide what has been promised.

ps_testWare

Software Testing Services

Tiensesteenweg 329
B-3010 Leuven
Tel.: +32 (16) 35.93.80
Fax: +32 (16) 35.93.88
e-mail: ps_testware@compuserve.com
<http://www.pstestware.com>



Copyright © 1999 ps_testware - Geert Pinxten - Test Automation - 37

toc
Intro
Start
Phase II
Phase III
TB
Less.
Info



Improve your **Processes**

— **Optimise your Business**

by

→ **Managing your Risks**

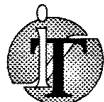
1

RiskDRIVER Project Risk Management

Quality Week Europe '99

What is RiskDRIVER?

A European initiative launched by a consortium of international partners, **RiskDRIVER** aims to promote **best practices in risk management for projects and process improvement.**



2

RiskDRIVER Project Risk Management

Quality Week Europe '99

Risk Management Process (ISO/IEC 15504)

ISO/IEC TR 15504 Part 2

A reference model for processes and process capability

Management process category (MAN)

- MAN.1 Management Process
- MAN.2 Project Management Process
- MAN.3 Quality Management Process



- **MAN.4 Risk Management Process**

The purpose of the Risk management process is to identify and mitigate the **project risks** continuously throughout the life-cycle of a project. The process involves establishing a focus on monitoring of risks at both the project and organisational levels.

3

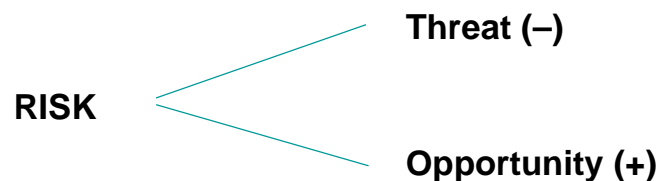
RiskDRIVER Project Risk Management

Quality Week Europe '99

What is Project Risk?

A project risk is an uncertainty that may impact the execution or outcome of the project.

Risk is measured in terms of *cost* (its effect on the project budget) or *time* (its effect on the project schedule). It may be negative - a *threat* risk - or positive - an *opportunity* risk.



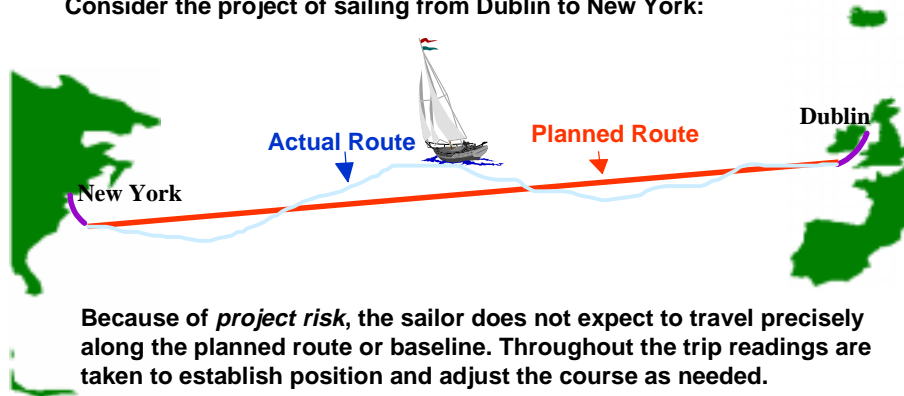
4

RiskDRIVER Project Risk Management

Quality Week Europe '99

The Project Plan is what you're NOT going to do!

Consider the project of sailing from Dublin to New York:



Because of *project risk*, the sailor does not expect to travel precisely along the planned route or baseline. Throughout the trip readings are taken to establish position and adjust the course as needed.

Risk management is not just contingency planning for when things go wrong, it is inextricably bound to the project plan which must be constantly revised.

5

RiskDRIVER Project Risk Management

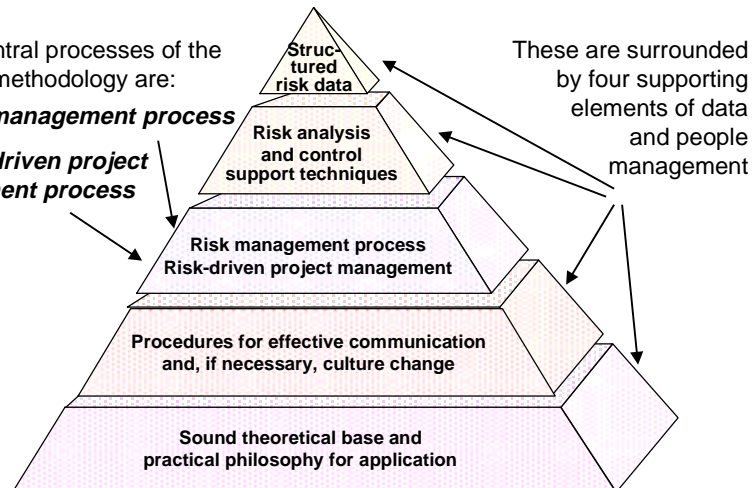
Quality Week Europe '99

RISKMAN

The European Project Risk Management Methodology

The two central processes of the RISKMAN methodology are:

- *The risk management process*
- *The risk-driven project management process*



6

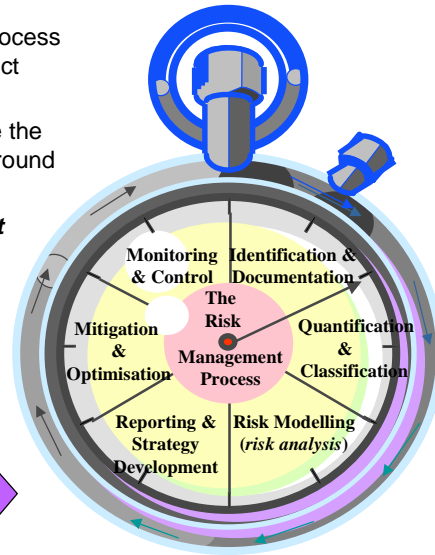
RiskDRIVER Project Risk Management

Quality Week Europe '99

The One-Minute Project Risk Manager

Project risk management is a cyclical process within the framework of risk-driven project management.

- The **risk management process** is like the second hand on a stopwatch, going around several times as the project proceeds.
- The **risk-driven project management process** represents a sequence of stages that runs its course just once through the life-cycle of the project, like the minute hand of the stopwatch.



7

RiskDRIVER Project Risk Management

Quality Week Europe '99

5-Step Risk Management Process



8

RiskDRIVER Project Risk Management

Quality Week Europe '99

Risk Identification

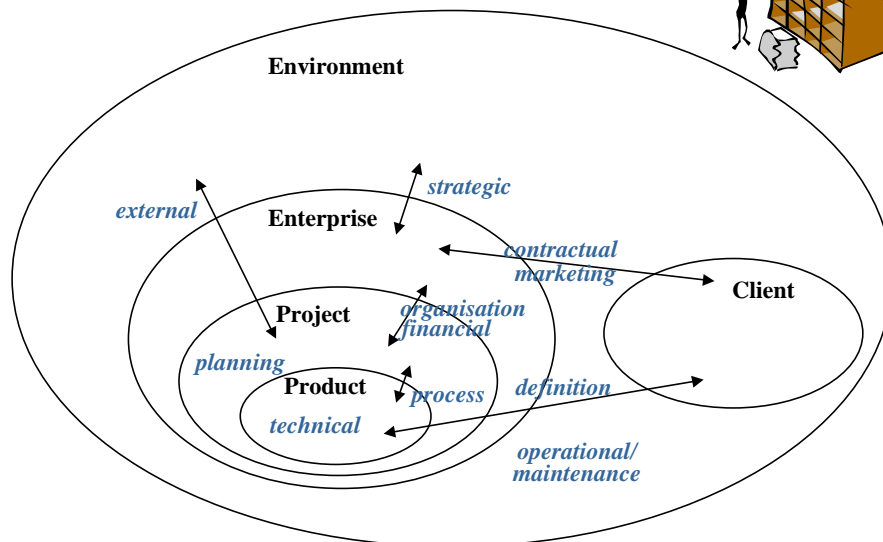
- Existing project documentation
- Structured interviews
- Company experts or external consultants
- Brainstorming sessions
- Standard questionnaires and checklists
- Risk databases and expert systems
- Methodological approaches

9

RiskDRIVER Project Risk Management

Quality Week Europe '99

Classification of Risk



10

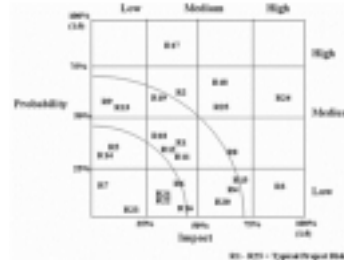
RiskDRIVER Project Risk Management

Quality Week Europe '99

Risk Assessment

- Estimate impact and probability of each risk
- Assign weighting to each risk based on risk exposure

Risk Exposure is the relative value that can be ascribed to any one risk, or to the sum of the risks faced. It is used to justify expenditures on mitigation, or on contingencies for risks that cannot be justifiably mitigated.



- Prioritise risks according to category, classification, weighting, imminence

11

RiskDRIVER Project Risk Management

Quality Week Europe '99

Risk Action Plan

Risk Mitigation

- Action to reduce, eliminate or avert the impact or probability of risks
- Intended for risks assessed with medium to high **exposure** or those rated as unacceptable or critical risks
- Requires up-front investment of time and money that cannot be recovered later if the risk does not occur

Contingency

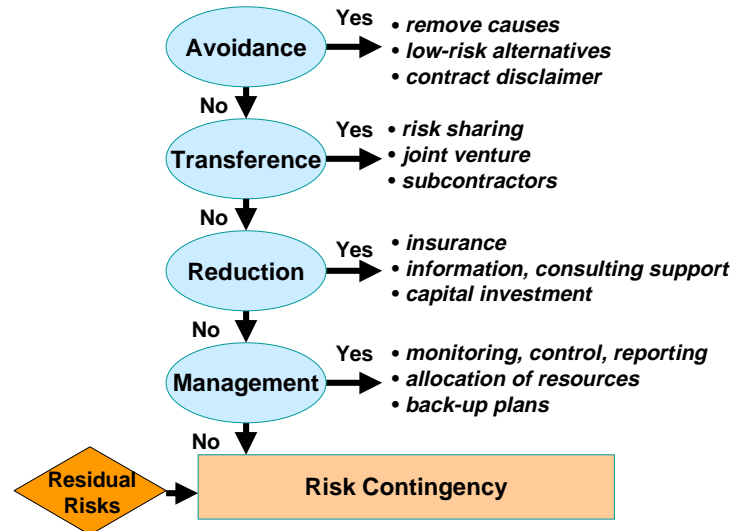
- Provision of time and/or funds to be used in the future should the risk actually materialise
- Covers risks that are assessed to be of a low likelihood and impact, and **residual risks** that have not been revealed during the identification process
- Can be recovered or reallocated if the risk does not occur

12

RiskDRIVER Project Risk Management

Quality Week Europe '99

Mitigation Paths



13

RiskDRIVER Project Risk Management

Quality Week Europe '99

Knowledge Capitalisation: The RiskDRIVER website

www.riskdriver.com is a risk management portal site offering free access to the following resources:

- ◆ Online and downloadable Risk Catalogues
- ◆ RiskProbe self-assessment tool
- ◆ Compiled best practices
- ◆ Discussion forums
- ◆ Case studies
- ◆ Risk management tool directory
- ◆ News, events, links to other relevant sites

14

RiskDRIVER Project Risk Management

Quality Week Europe '99

www.riskdriver.com

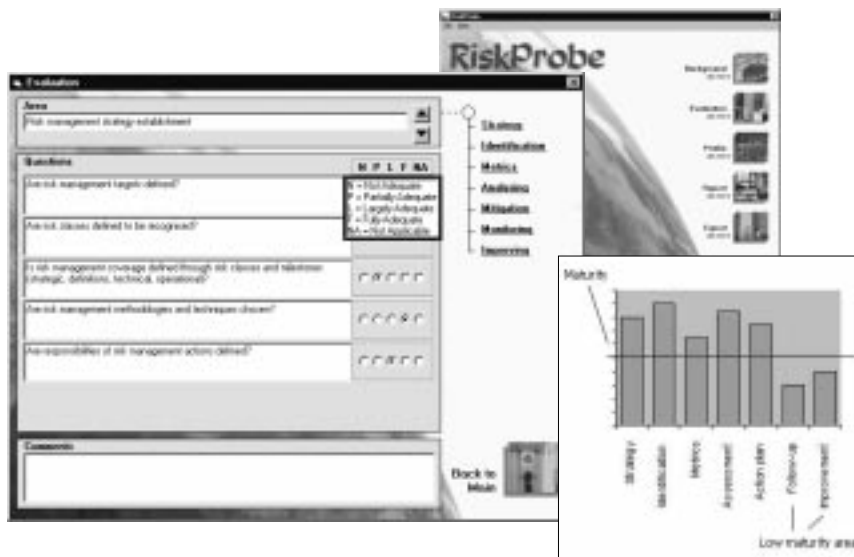


15

RiskDRIVER Project Risk Management

Quality Week Europe '99

Self-Assessment Tool



16

RiskDRIVER Project Risk Management

Quality Week Europe '99

Risk Catalogue

Risk Catalogues

Please select an option on the bar below

- Risk Catalogue
- Risky
- Mitigation Actions
- Help

Risk Form

Current Filter: None

Name: Final management request to change

Substance: The current state for change risks are not transparent, tracked and planned, so we need to construct requirements management to not lose customer

Classification: Contract

Rules: Management

State: Software

Impact Description: There have a contract to implement the changes. The contract requires to be done and user should be do program at their own money (customer is contract)

Management Strategy:

Actions

| # | Name | Description |
|---|--------------------------------|---|
| 1 | Identify requests for change | Identify request for change to the basic state... |
| 2 | Manage requests | When defined and accepted by sales, analysis... |
| 3 | State early about user involve | Consequently, it is essential for participants... |

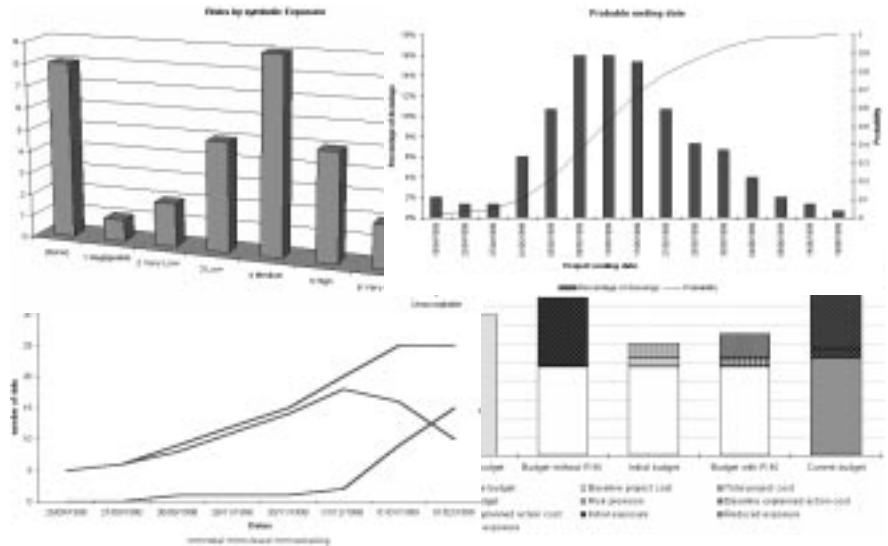
Page 1

17

RiskDRIVER Project Risk Management

Quality Week Europe '99

RISKMAN Software

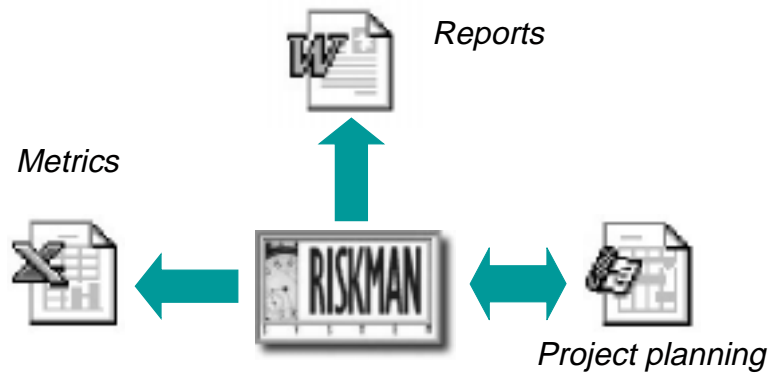


18

RiskDRIVER Project Risk Management

Quality Week Europe '99

RISKMAN Software Integration with Office Applications



19

RiskDRIVER Project Risk Management

Quality Week Europe '99

The RiskDRIVER Consortium



- Sylvain Schieber *CR2A-DI (France)*
- Stefano Allari *IBK (Germany)*
- Miklos Biro *IT (Hungary)*
- Jouko Luukas *CCC Software Professionals (Finland)*
- Larry Moffett *e-Strategy (Belgium)*
- Jose Maria Sanz *European Software Institute (Spain)*
- James Stein *OPL UK (UK)*
- David Storch *RADICALmedia (Portugal)*

Supported by the



20

RiskDRIVER Project Risk Management

Quality Week Europe '99



Lessons Learned in the Real World

VT7 - Thursday November 4th at 11:00

David Eade
McCabe & Associates

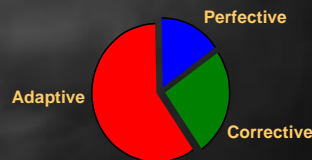
QWE'99

Software Development

- **Waterfall**
 - Traditional Model
 - V Model
- **Spiral**
 - Evolutionary
 - RAD
- **Adaptive / Extreme Etc.**

Software Maintenance

- **Perfective**
 - Performance/Behaviour
 - **Corrective**
 - Bug Fixes
 - **Adaptive**
 - New Functionality
- Code Modification
- On-Going Development



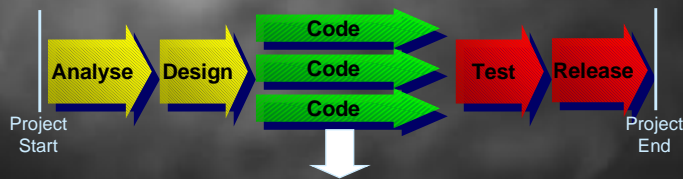
Waterfall / Traditional / V

Challenges

Multiple Programmers

Issues

Code Quality
Developer Testing
Code Reviews



Quality: Coding Standards
Different Testing Practices
Code Reviews

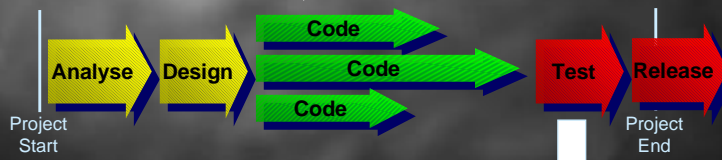
Waterfall / Traditional / V

Challenges

Multiple Programmers
Variable Workloads

Issues

Code Quality
Developer Testing
Code Reviews
Testing Time
Defect Reduction
Timescales



TimeScales: Reduced Testing Time
Increased Defects



5

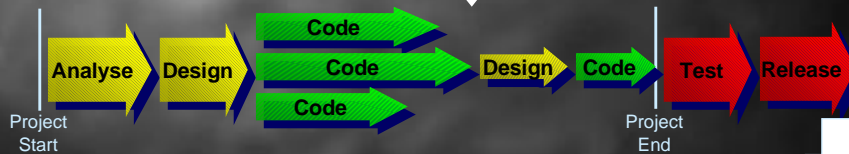
Waterfall / Traditional / V

Challenges

Multiple Programmers
Variable Workloads
New Requirements
Perfective Enhancements

Issues

Code Quality
Developer Testing
Code Reviews
Testing Time+
Defect Reduction+
Timescales+



Late Delivery
Reduced Testing Time
Increased Defects



6

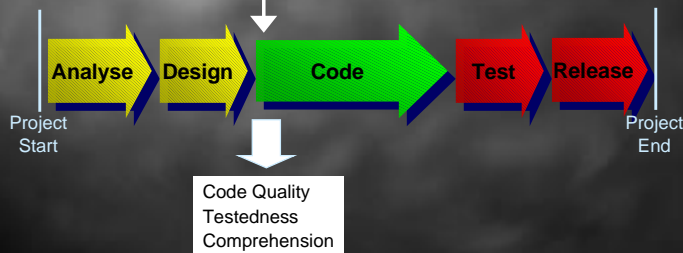
Waterfall / Traditional / V

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use

Issues

- Code Quality+
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction+
- Timescales+
- Testedness
- Comprehension



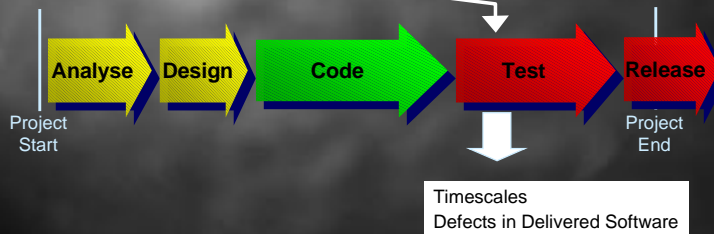
Waterfall / Traditional / V

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality

Issues

- Code Quality+
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension



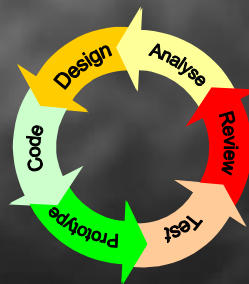
Spiral / Evolutionary / RAD

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality

Issues

- Code Quality+
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension



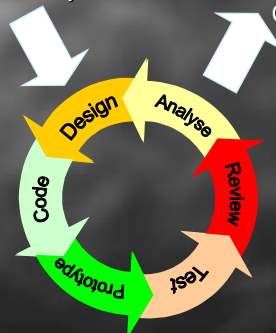
Spiral / Evolutionary / RAD

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality

Issues

- Code Quality+
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension



Spiral / Evolutionary / RAD

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality

Issues

- Code Quality++
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension+
- Metrics Trending



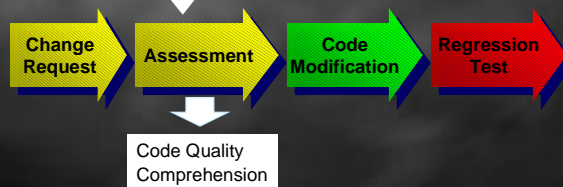
Maintenance

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality
- Impact of Change

Issues

- Code Quality+++
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension++
- Metrics Trending



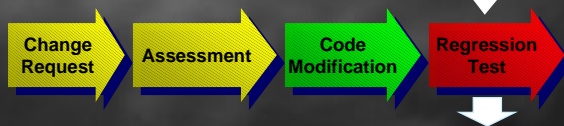
Maintenance

Challenges

Multiple Programmers
Variable Workloads
New Requirements
Perfective Enhancements
Code Re-Use
Poorly Defined End-Point for Testing
Ever-Increasing Functionality
Impact of Change
Testing Modified Code

Issues

Code Quality+++
Developer Testing
Code Reviews
Testing Time+
Defect Reduction++
Timescales++
Testedness+
Comprehension++
Metrics Trending
Coverage of Changed Code
Tracking Changes



Coverage of Changed Code
Tracking Changes

Issues

Code Quality
Developer Testing
Code Reviews
Testing Time
Defect Reduction
Timescales
Testedness
Comprehension
Metrics Trending
Coverage of Changed Code
Tracking Changes



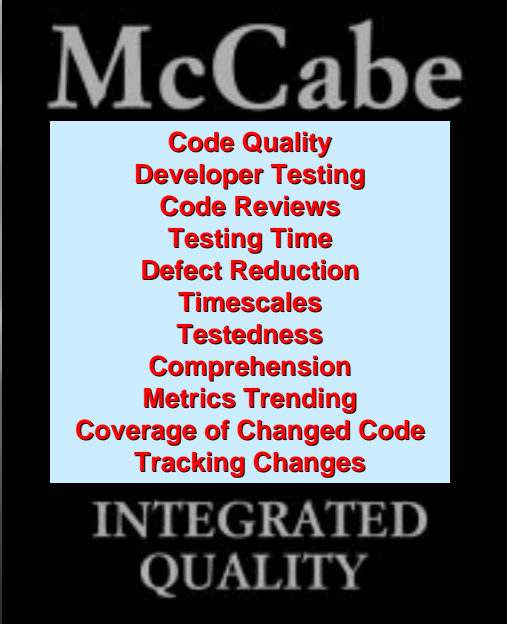
McCabe

- Code Quality
- Developer Testing
- Code Reviews
- Testing Time
- Defect Reduction
- Timescales
- Testedness
- Comprehension
- Metrics Trending
- Coverage of Changed Code
- Tracking Changes

**INTEGRATED
QUALITY**

 McCabe Associates

15



McCabe

- Code Quality
- Developer Testing
- Code Reviews
- Testing Time
- Defect Reduction
- Timescales
- Testedness
- Comprehension
- Metrics Trending
- Coverage of Changed Code
- Tracking Changes

**INTEGRATED
QUALITY**

McCabe QA

McCabe Change

McCabe Compare


McCabe Data

McCabe Test

McCabe TestCompress

McCabe Slice

McCabe ReTest

 McCabe Associates

16

McCabe QA

Code Quality
Code Reviews
Defect Reduction
Comprehension
Metrics Trending

Demonstration

McCabe IQ
McCabe QA
McCabe Reengineer

McCabe Test

Developer Testing
Testing Time
Timescales
Testedness
Coverage of Changed Code

Demonstration

McCabe IQ
McCabe Test
McCabe Reengineer



Lessons Learned in the Real World

VT7 - Thursday November 4th at 11:00

David Eade
McCabe & Associates

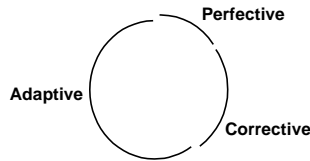
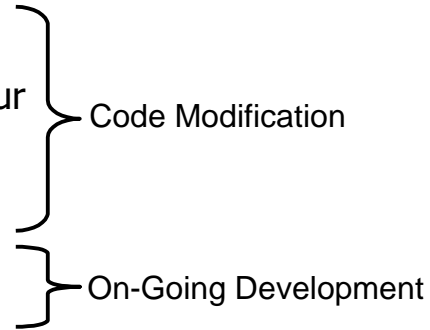
QWE'99

Software Development

- Waterfall
 - Traditional Model
 - V Model
- Spiral
 - Evolutionary
 - RAD
- Adaptive / Extreme Etc.

Software Maintenance

- Perfective
 - Performance/Behaviour
- Corrective
 - Bug Fixes
- Adaptive
 - New Functionality



3

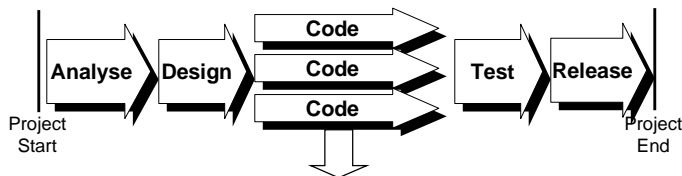
Waterfall / Traditional / V

Challenges

Multiple Programmers

Issues

Code Quality
Developer Testing
Code Reviews



Quality: Coding Standards
Different Testing Practices
Code Reviews



4

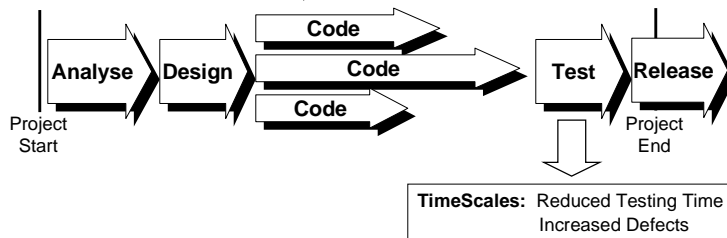
Waterfall / Traditional / V

Challenges

Multiple Programmers
Variable Workloads

Issues

Code Quality
Developer Testing
Code Reviews
Testing Time
Defect Reduction
Timescales



McCabe Associates

5

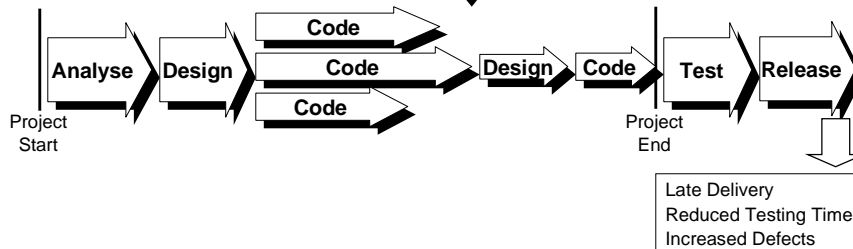
Waterfall / Traditional / V

Challenges

Multiple Programmers
Variable Workloads
New Requirements
Perfective Enhancements

Issues

Code Quality
Developer Testing
Code Reviews
Testing Time+
Defect Reduction+
Timescales+



McCabe Associates

6

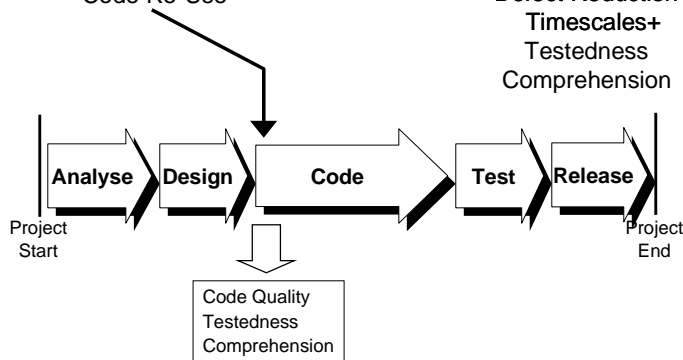
Waterfall / Traditional / V

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use

Issues

- Code Quality+
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction+
- Timescales+
- Testedness
- Comprehension



7

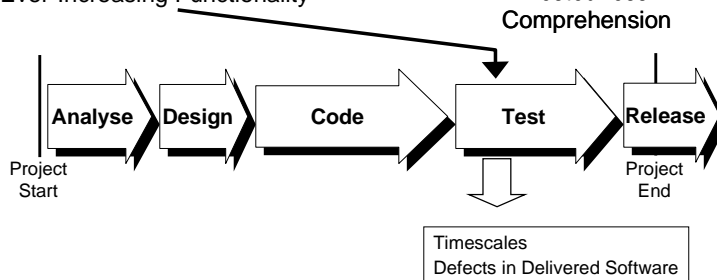
Waterfall / Traditional / V

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality

Issues

- Code Quality+
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension



8

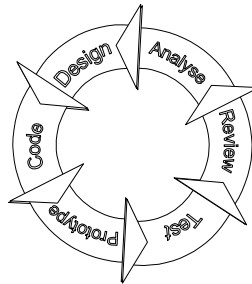
Spiral / Evolutionary / RAD

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality

Issues

- Code Quality+
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension



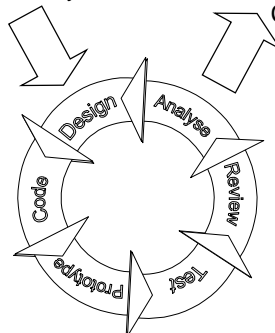
Spiral / Evolutionary / RAD

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality

Issues

- Code Quality+
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension



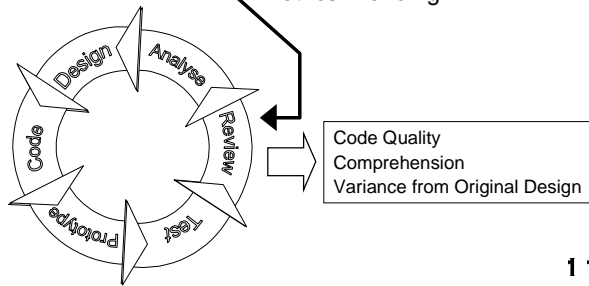
Spiral / Evolutionary / RAD

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality

Issues

- Code Quality++
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension+
- Metrics Trending



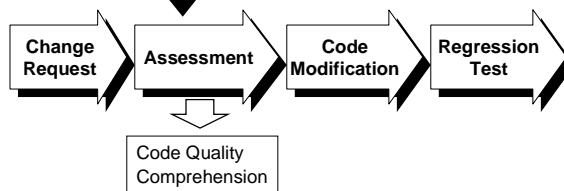
Maintenance

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality
- Impact of Change

Issues

- Code Quality+++
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness
- Comprehension++
- Metrics Trending



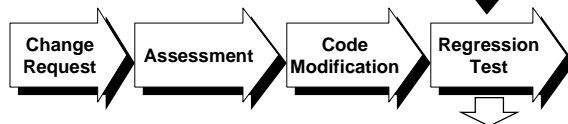
Maintenance

Challenges

- Multiple Programmers
- Variable Workloads
- New Requirements
- Perfective Enhancements
- Code Re-Use
- Poorly Defined End-Point for Testing
- Ever-Increasing Functionality
- Impact of Change
- Testing Modified Code

Issues

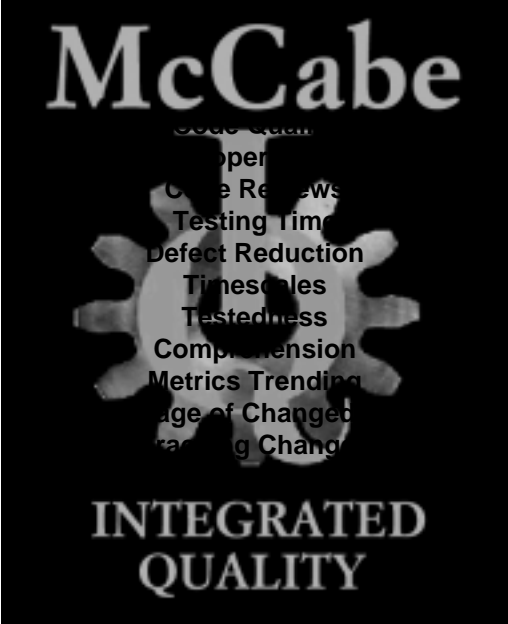
- Code Quality+++
- Developer Testing
- Code Reviews
- Testing Time+
- Defect Reduction++
- Timescales++
- Testedness+
- Comprehension++
- Metrics Trending
- Coverage of Changed Code
- Tracking Changes




Coverage of Changed Code
Tracking Changes

Issues

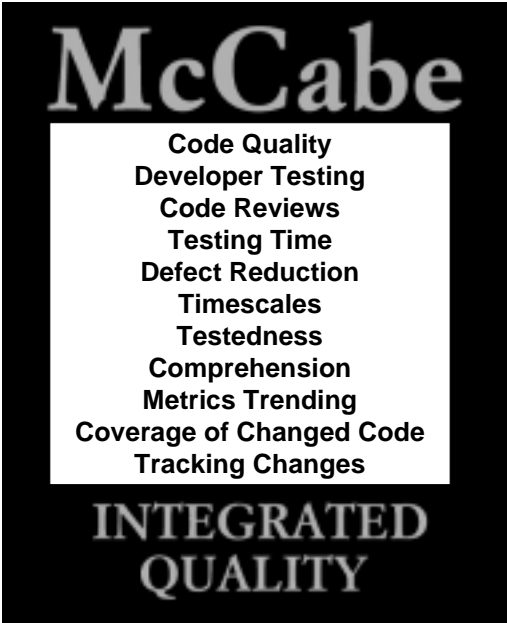
- Code Quality
- Developer Testing
- Code Reviews
- Testing Time
- Defect Reduction
- Timescales
- Testedness
- Comprehension
- Metrics Trending
- Coverage of Changed Code
- Tracking Changes



McCabe
 Code Quality
 Developer Testing
 Code Reviews
 Testing Time
 Defect Reduction
 Timescales
 Testedness
 Comprehension
 Metrics Trending
 Coverage of Changed Code
 Tracking Changes
**INTEGRATED
 QUALITY**

 McCabe Associates

15



McCabe
 Code Quality
 Developer Testing
 Code Reviews
 Testing Time
 Defect Reduction
 Timescales
 Testedness
 Comprehension
 Metrics Trending
 Coverage of Changed Code
 Tracking Changes
**INTEGRATED
 QUALITY**

McCabe QA

McCabe Change

McCabe Compare


McCabe Data

McCabe Test

McCabe TestCompress

McCabe Slice

McCabe ReTest

 McCabe Associates

16

McCabe QA

Code Quality
Code Reviews
Defect Reduction
Comprehension
Metrics Trending

Demonstration

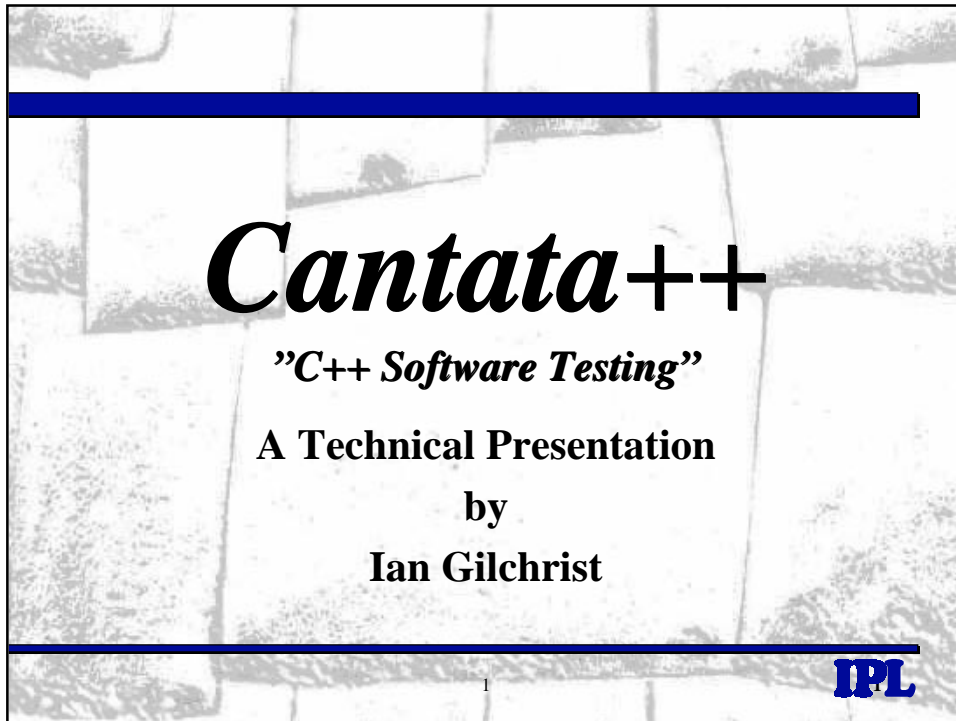
McCabe IQ
McCabe QA
McCabe Reengineer

McCabe Test

Developer Testing
Testing Time
Timescales
Testedness
Coverage of Changed Code

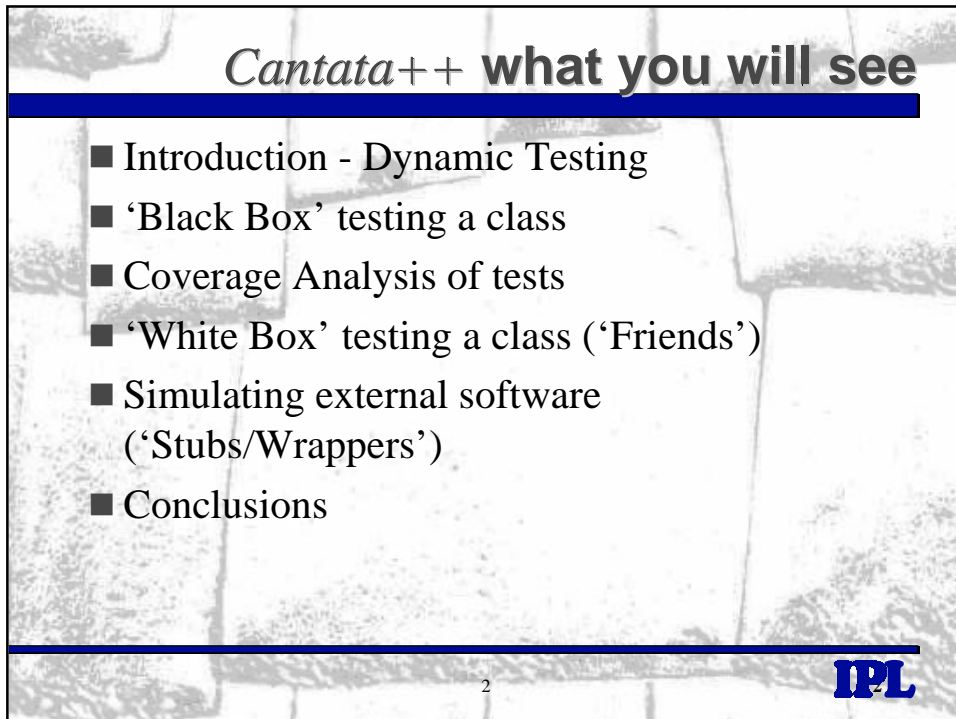
Demonstration

McCabe IQ
McCabe Test
McCabe Reengineer



Cantata++
”C++ Software Testing”
A Technical Presentation
by
Ian Gilchrist

1 **IPL**



***Cantata++* what you will see**

- Introduction - Dynamic Testing
- ‘Black Box’ testing a class
- Coverage Analysis of tests
- ‘White Box’ testing a class (‘Friends’)
- Simulating external software (‘Stubs/Wrappers’)
- Conclusions

2 **IPL**

Who is it for?

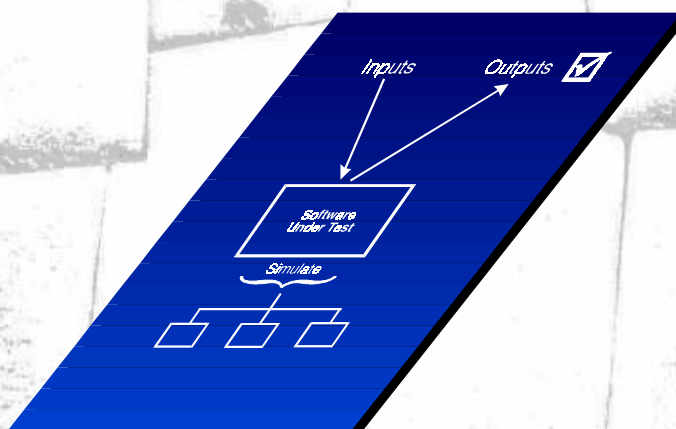
Who is *Cantata++* for?

- Developers of 'high-reliability' software, using C++
 - improve reliability by fully testing software at the component level
- People who want to do coverage analysis on C++ systems
 - discover weaknesses in their systems tests, and thus hopefully improve these

3

IPL

Dynamic Testing



4

IPL

Dynamic Testing

- Dynamic testing is the most fundamental form of verification
 - Check that the software behaves according to its specification - “Does it work?”
- *Cantata++* makes dynamic testing
 - REPEATABLE and AUDITABLE
 - AUTOMATED - hence fast
 - REUSABLE - contributes to cost-effectiveness

5

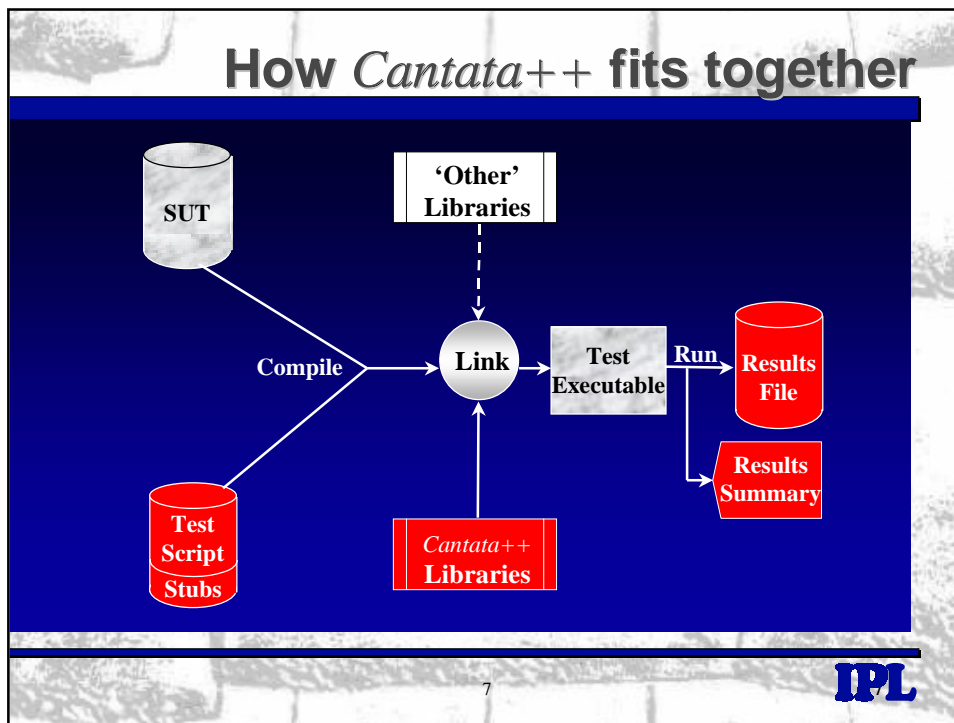
IPL

Dynamic Testing Techniques

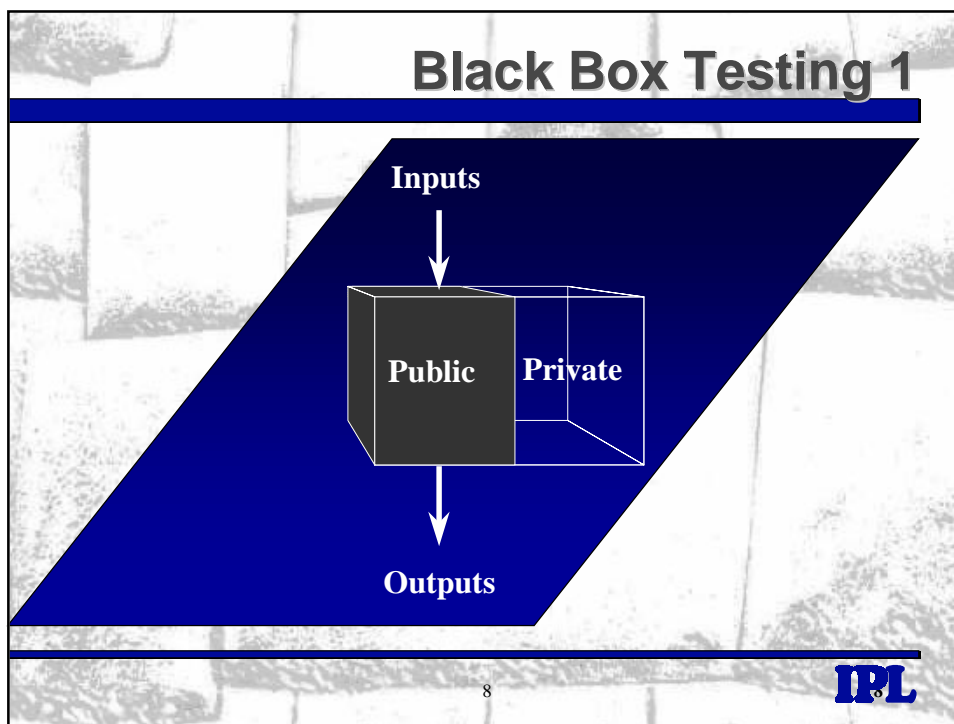
- *Cantata++* supports:
 - UNIT and INTEGRATION testing
 - BLACK BOX and WHITE BOX testing
 - SIMULATION of external classes/calls
 - REGRESSION testing
 - TIMING analysis

6

IPL



7



8

Black Box Testing 2

- Black box testing means testing a class **purely by its public interface.**
- Call class methods in a planned sequence, passing in known values and setting checks on return values.
- Test script written in C++ using the *Cantata++* Test Harness (CPPTH).

9

IPL

Black Box Testing - Demo 1

Class **Stack** has the public interface:

- > **'constructor'**
- > **is_empty**
- > **push** (with **new_error** exception)
- > **pop** (with **empty_pop** exception)
- > **'copy constructor'**
- > **'destructor'**

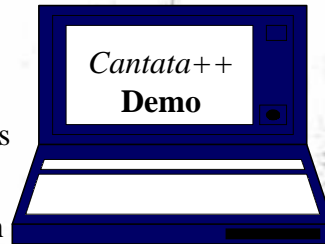
Private component is pointer to first node in linked list.

10

IPL

Black Box Testing - Demo 2

- Two test cases to begin with:
 - **“empty stack”**
 - ‘is_empty’ should be true
 - ‘pop’ should throw ‘empty_pop’ exception
 - **“push/pop”**
 - ‘push’ some known numbers
 - ‘is_empty’ should be false
 - ‘pop’ numbers and check values
 - ‘is_empty’ should be true
 - no exceptions should be thrown



11

IPL

Black Box Testing - Demo 3

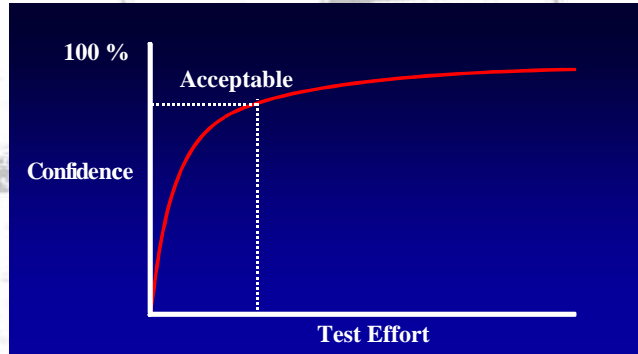
What happened?

- ‘empty class’ seems to work
 - ‘push’ contained a bug
 - Inspection of code revealed probable cause of the bug
 - Code was changed and the retest passed!
- So can we conclude the code is ready to use?
Have we finished testing?

12

IPL

How Much Testing ?

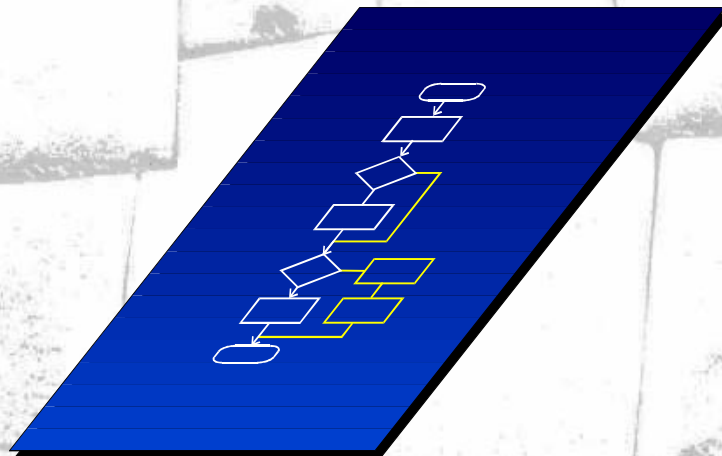


- An 'acceptable' level of confidence is achievable, but needs to be set in objective terms using **coverage analysis**.

13

IPL

Coverage Analysis



14

IPL

Coverage Analysis - Why ?

- If the purpose of software testing is to gain **“An acceptable level of confidence in the software...”**, then this needs to be measured.
- Coverage Analysis is the objective measurement of acceptability, against which the effectiveness of test effort can be judged.
- Testing can then be controlled **“... at an acceptable level of cost”**.

15

IPL

Coverage Analysis - When?

- Builds upon Dynamic Testing
 - Measure the “amount” of the software under test which has been exercised
 - The more coverage the more confidence
 - Lots of definitions of “amount”!
- Can be used with Unit, Integration or System level testing
- Coverage can be integrated with *Cantata++* dynamic testing or ‘standalone’

16

IPL

Coverage Analysis - What?

- Structural coverage metrics
 - entrypoint, statements, decisions, boolean operators etc.
- OO coverage metrics
 - coverage of inheritance
 - template instantiations
- Other Metrics
 - “masking” MC/DC
 - boundary (relational) coverage

17

IPL

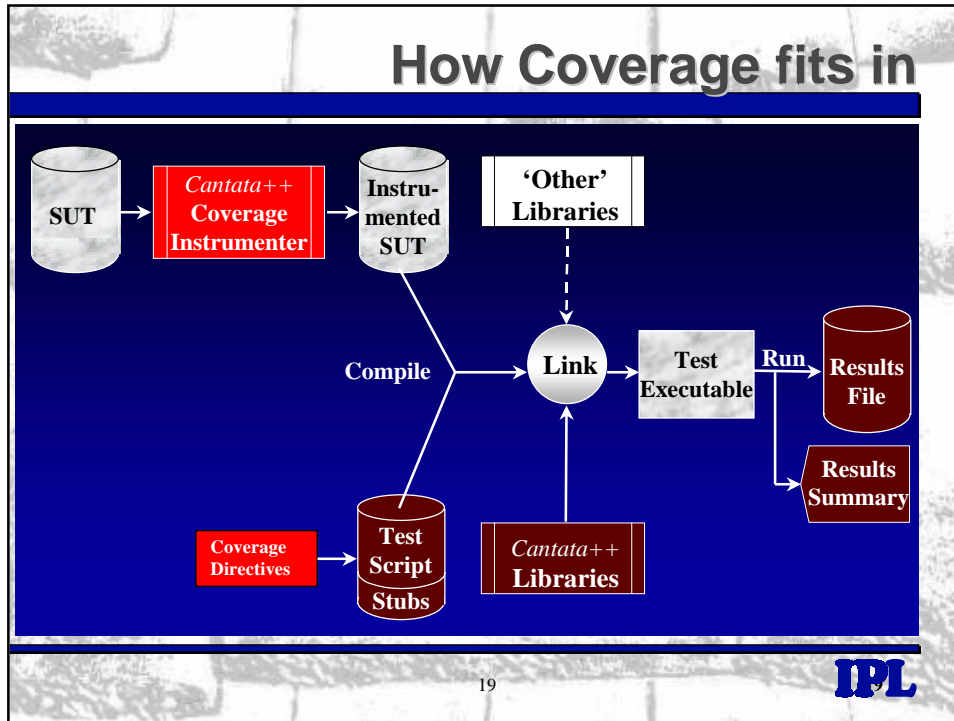
Coverage Analysis - How?

Coverage analysis is carried out by:

- Instrumenting code under test
 - enables coverage by inserting software ‘probes’
- Adding *Cantata++* Analysis to the test script
 - checks and reports (textual and graphic)
- Re-running test and look at results:
 - coverage check - Pass, Fail or Warning?
 - coverage diagnostics show where code not executed

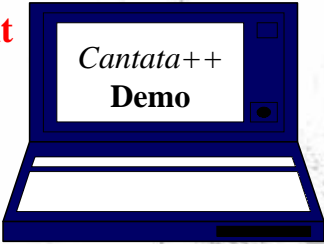
18

IPL




Coverage Analysis - Demo

- Back to the 'stack' class
- Set up up coverage analysis to check
 - Pass/Fail on 100% **Entrypoint** coverage
 - (I.e. all methods called)
 - Warnings on 100% **Statement** and **Decision** coverage



20



Coverage Analysis - Demo 2

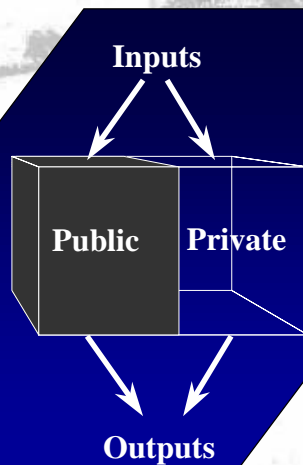
What happened?

- Entrypoint coverage failed with < 100%
 - Showed that the copy constructor had not been executed - we need a new test case.
- Statement and Decision coverage produced warnings with < 100%
 - We will deal with these later

21

IPL

White Box Testing 1



22

IPL

White Box Testing 2

- White box testing means testing a class with **access to the private implementation details.**
- Uses the 'Friends' mechanism.
 - Testability Instrumentation is run on class under test to declare that the test class is a 'friend' of the class under test.
 - This is completely automatic.

23

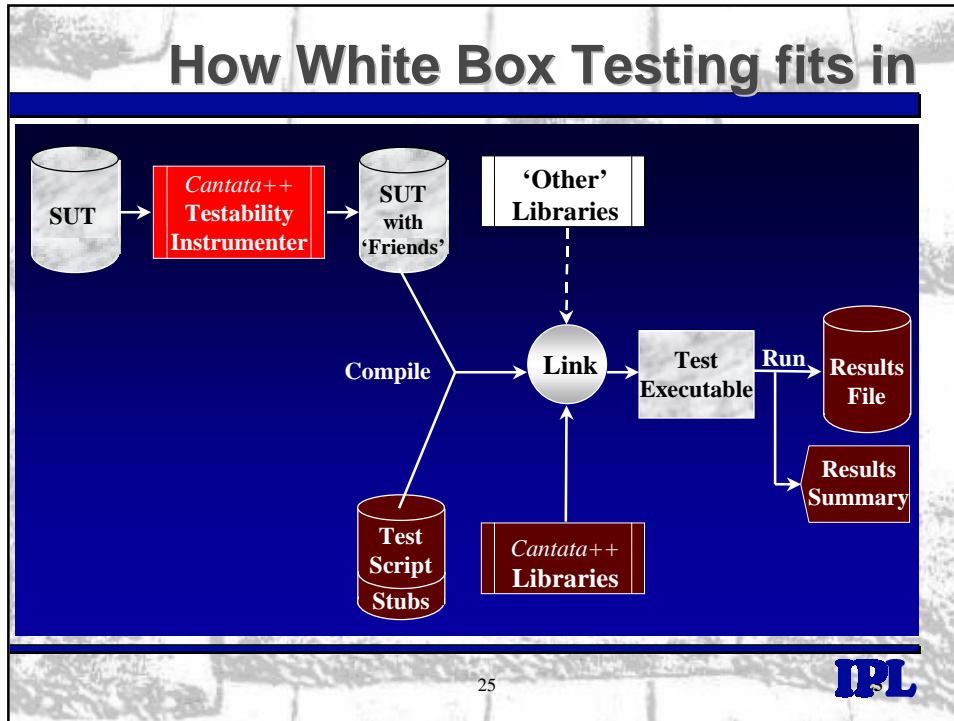
IPL

White Box Testing 3

- 'Friends' allows test to :
 - call private methods directly
 - gain a high level of coverage more easily
 - set and check private data directly
 - more direct so more efficient testing
 - better at finding obscure implementation faults such as pointers left corrupted

24

IPL



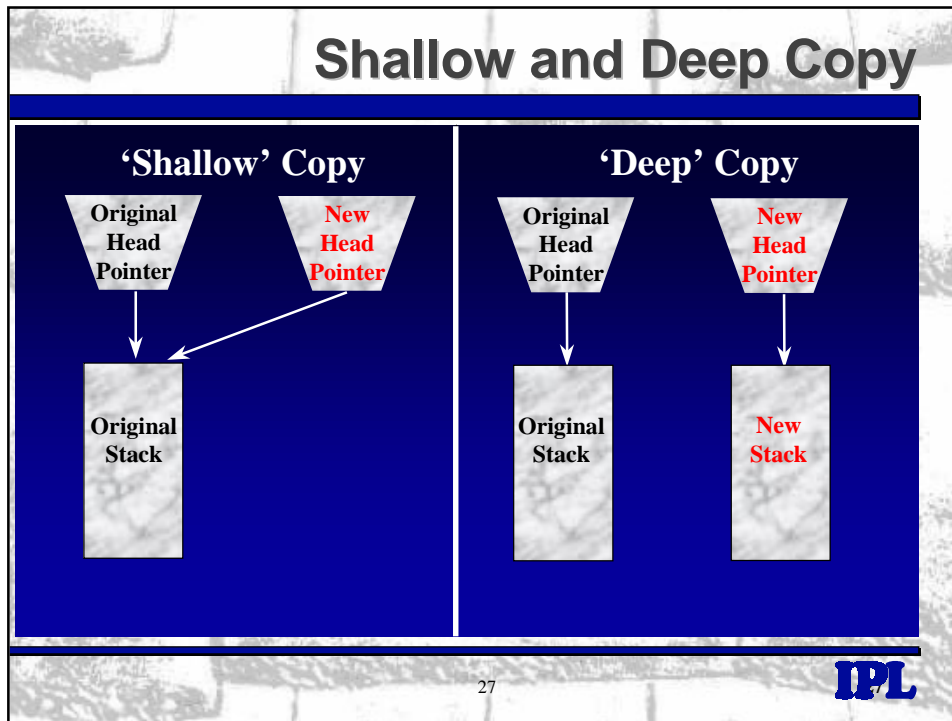
White Box Testing - Demo 1

- Back to 'stack' class
- Want to do a test on the copy constructor, and ensure that 'deep copy' has taken place.
- Use Testability Instrumentation to add 'friend' to class under test.
- Add new test case to test script.

An illustration of a laptop computer. The screen displays the text 'Cantata++ Demo'. The laptop is shown from a slightly elevated perspective.

26

IPL




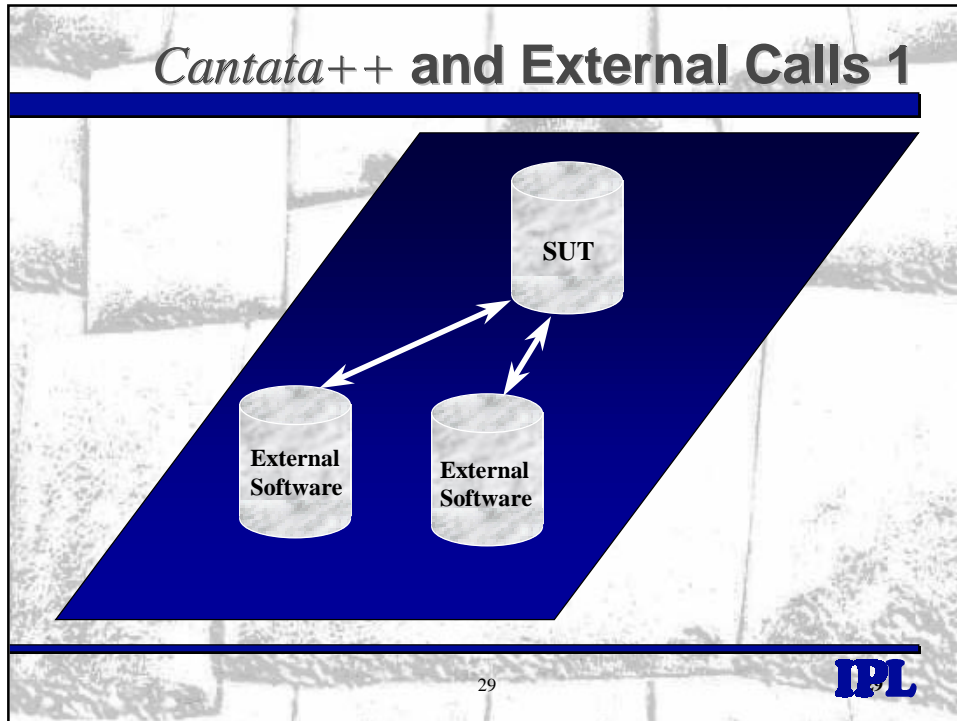
White Box Testing - Demo 2

What happened?

- The Black Box checks indicated that the copy had worked as it should.
- The White Box check however showed that only a shallow copy had taken place.
- Code was corrected.
- The test was forced by Coverage Analysis and enabled by the 'Friend' technique.
- Coverage 'Warning' for an external call.

28

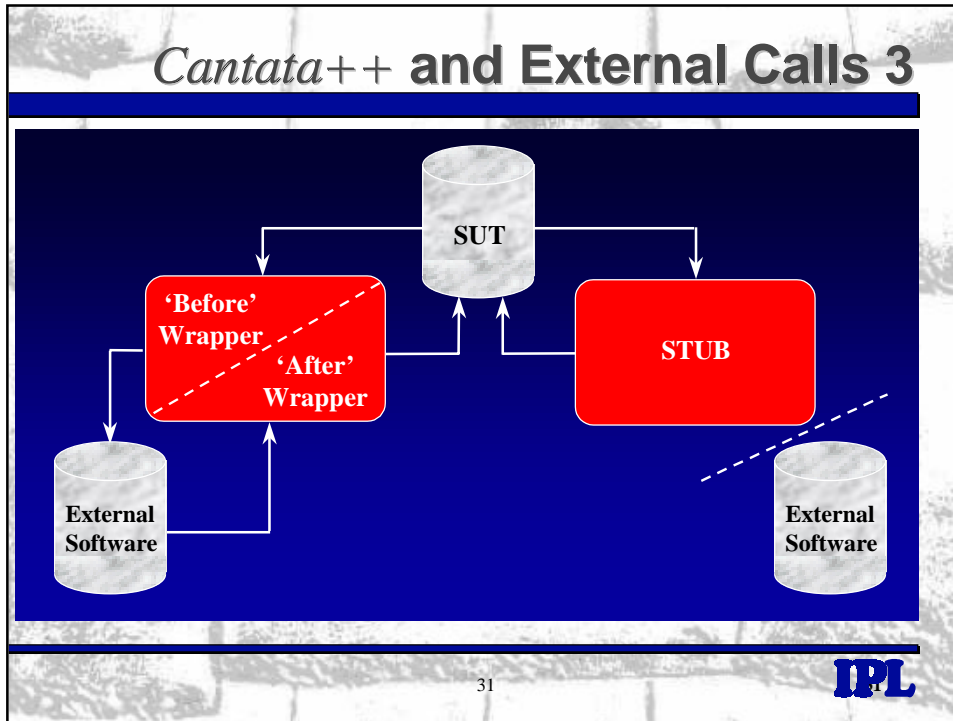




Cantata++ and External Calls 2

- External calls can be treated in two different ways with *Cantata++*
 - **STUBS** - replace 'real' function with a local substitute
 - works very well with function calls
 - **WRAPPERS** - intercept call to real external software
 - intended to solve the C++ 'stubbing problem'


30



Stubbing or Wrapping

| | Stubbing | Wrapping |
|------------------------------------|--------------|--------------|
| Check call order | ✓ | ✓ (optional) |
| Check parameters | ✓ (optional) | ✓ (optional) |
| Call original function | ✗ | ✓ |
| Set return value | ✓ | ✓ (optional) |
| Throw exception | ✓ (optional) | ✓ (optional) |
| Change output parameters | ✓ (optional) | ✓ (optional) |
| Call original with modified params | ✗ | ✓ |
| Use with system calls | ✗ | ✓ |
| Use selectively | ✗ | ✓ |
| Original function linked with test | ✗ | ✓ |

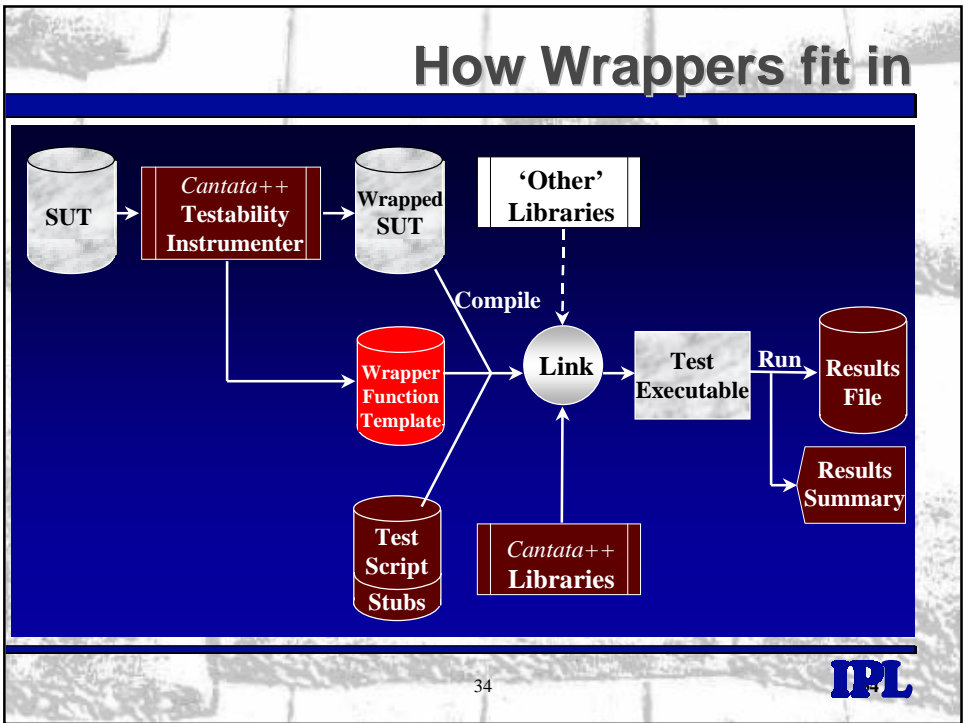

32



Cantata++ and External Calls 6

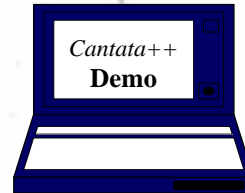
- When do we use wrappers?
 - Example - software under test accesses database
 - Uses DBQuery class
 - DBQuery is provided as a library so cannot change source
 - DBQuery constructors cannot be stubbed
 - Want to check that correct calls have been made
 - Check correct parameters passed
 - Perhaps check order calls made ...
 - ... but sometimes we don't care as long as they happen
 - Sometimes force return values

33



Wrappers - Demo

- Back to 'stack' class demo
- Coverage has shown lack of completeness in the area of 'new' error response. What to do?
- Answer - wrap the call to 'new'!
 - intercept the AFTER call and change the return value to Null.
 - Does the software do what it should do:
(throw the new_error exception)?



35

IPL

Demo - Summary

- This demo has been conducted in five parts:
 - Simple **Black Box** tests - found a bug, and **Coverage Analysis** showed some gaps in the testing:
 - 'Copy' method not called
 - The error branch of 'new' in push not executed.
 - A **White Box** test (enabled through the 'Friend' technique) showed that the 'copy' method contained a serious bug.
 - **Wrapping** 'new' allowed an error code to be returned, thus finishing the job.

36

IPL

Demo - not shown

- There has not been time or space to show:
 - ‘Reuse’ Testing - Derived Classes
 - Static Analysis
 - OO Coverage
 - Template testing

37

IPL

Read all about it!

The IPL paper “C++. It’s Testing, Jim, But Not As We Know It!” additionally describes:

- Design for Testability
 - Abstract Base Classes
- Test Case Reuse
 - Factory Classes
- Coverage in Context
 - Polymorphism
 - State-based coverage

38

IPL

Training from IPL

- Training for Cantata++ is offered as two one-day courses:
 - 'Introduction to Testing C++ with *Cantata++*'
 - 'Using *Cantata++*'
- Both days can be provided in classroom format ('public' or 'private').
- Both days are available on CD-ROM.

39

IPL

Conclusion

- Since its introduction C++ code has always been difficult to test. This has frequently been made an excuse to do no testing.
 - **Result - Test the application not the code, with consequent (very) extended system testing.**
- *Cantata++* makes dynamic testing of C++
 - REPEATBLE and AUDITABLE
 - AUTOMATED
 - REUSABLE

40

IPL

Cantata++

**Any more
questions?**

41

IPL

QWE'99

Automating Cleanroom Management



Ronnie Van Parijs
CA-Belgium
ronnie.vanparijs@CAI.COM

Automating Cleanroom Management

- ▶ What is Clean Room Management
- ▶ Change and Configuration Management Benefits
- ▶ Endeavor: The Solution



What is Cleanroom Management?

Cleanroom, as defined by IBM, is a “managerial and technical process for the development of software with ultra-high quality with certified reliability”

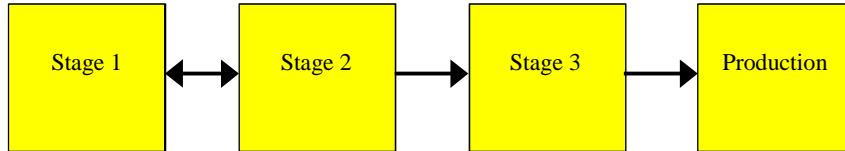
“Management of the Cleanroom process is based on an incremental life cycle in which development and certification are conducted in a pipeline of user-function increments.”

Incremental Life Cycle

- ▶ Composed of site-defined requirements executed in the system environment, designed to accumulate changes into a final product.
 - Integration is top-down and continuous
 - Implementation of process key to ensure change process is error free
 - Each change increment implemented onto of the last, following the same procedures

Incremental Life Cycle

Automation, the Key to Success



- ▶ Components of change increment move together
- ▶ Required processes and tests can be automated -- ensuring accuracy
- ▶ Audit trail of all activities

Change and Configuration Management

The ability to manage, control and improve the process of building, changing and operating software systems.

CCM Benefits

- ▶ Desired business standards are applied...everytime
- ▶ Audit requirements are achieved
- ▶ Concurrent development is under control
- ▶ Prior executables recreated quickly and correctly
- ▶ Accurate source to executables trace

CCM Benefits

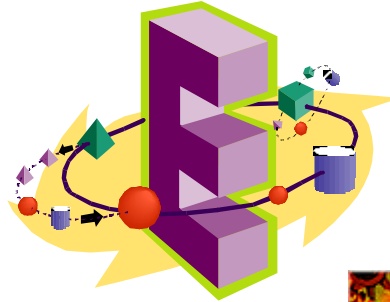
- ▶ Track all components of a change
- ▶ All required components are included
- ▶ Appropriate approvals performed in a timely fashion
- ▶ Reduced downtime
- ▶ Improve time to market



The Solution: Endeavor

Endeavor features:

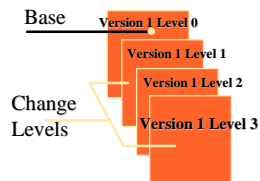
- ▶ Version control
- ▶ Change control
- ▶ Configuration control
- ▶ Life-cycle modeling
- ▶ Concurrency protection
- ▶ Auditing and reports



Version and Change Control

Standards Enforcement

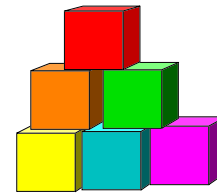
- ▶ Know who, when, why change made
- ▶ Ensure the right people get involved



Configuration Management

Standards Enforcement

- ▶ Creates executables when source is changed
 - Standards defined and enforced
 - Standard processes implemented
 - Repeatable procedures established



▶ Footprinting

- Prevents movement of executables into any environment without corresponding source code(s)



Life-cycle Modeling

Standards Enforcement

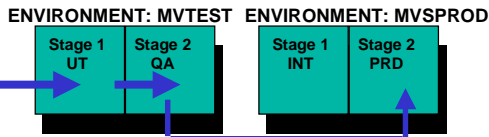
- ▶ Establish and enforce site-specific standards for life cycle administration
- ▶ For specific change increments, individual applications or across the organization
- ▶ Ensures process control

Life Cycle Modeling

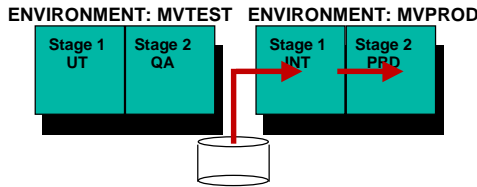
Map - Defines the path of migration

Route 1:

Standard Development

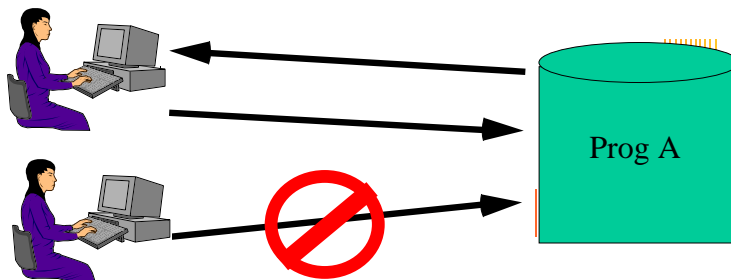


Route 2:
Emergency Fixes

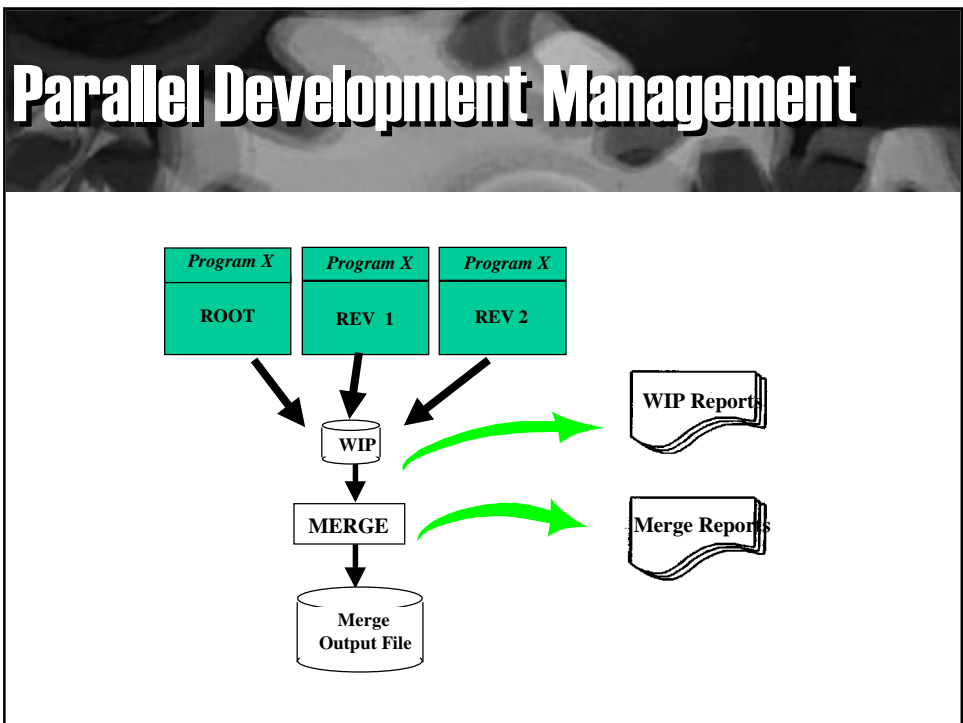
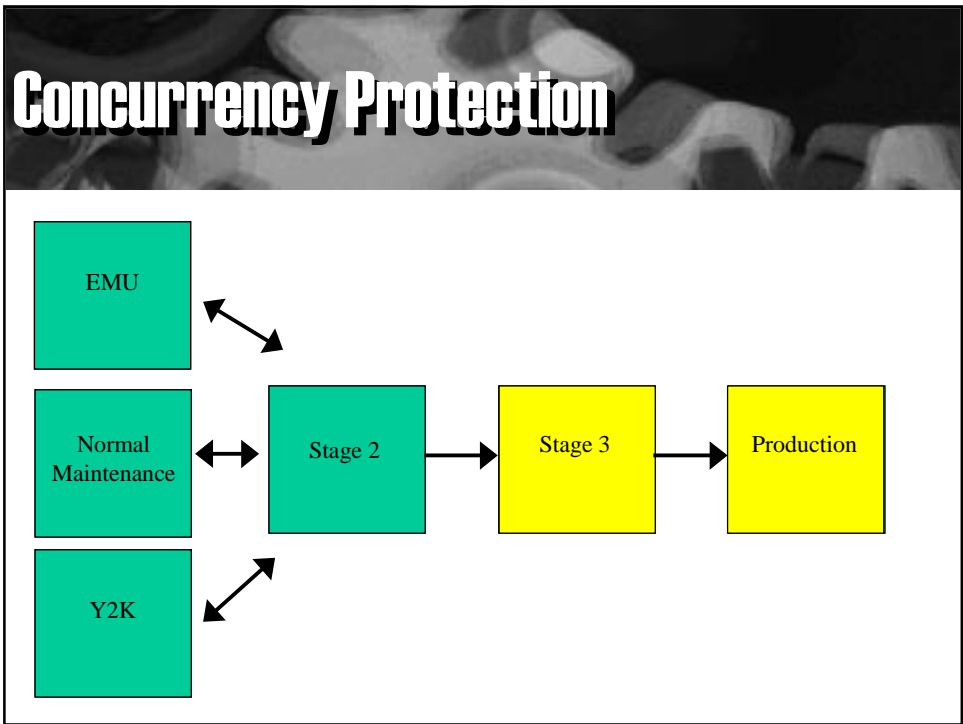


Concurrency Protection

Parallel Development Management



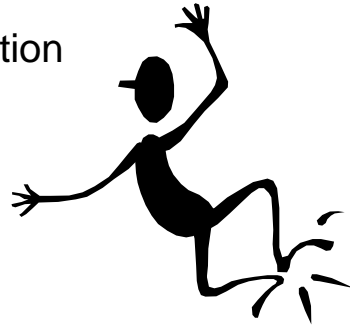
A sign-in-signout facility prevents unintentional overlaying of code



Auditing and Reporting

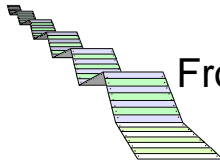
Use ad hoc or predefined reports with selection criteria

- Application logical classification
- Activity
- Application changes



Auditing and Reporting

Impact Analysis



From Synchronize Reports or



| |
|------------------|
| • Component |
| • Used by Report |
| • |
| • |
| • |
| • |
| • |
| • |

| |
|---------------------|
| • Component |
| • Where Used Report |
| • |
| • |
| • |
| • |
| • |
| • |

| |
|-----------------|
| • Related |
| • Entity Report |
| • |
| • |
| • |
| • |
| • |
| • |

What is Cleanroom Management?

Cleanroom, as defined by IBM, is a “managerial and technical process for the development of software with ultra-high quality with certified reliability”

“Management of the Cleanroom process is based on an incremental life cycle in which development and certification are conducted in a pipeline of user-function increments.”

Endevor - integration

Endevor for MVS

Endevor Workbench

Endevor Workstation



CCC/Harvest

One common user-interface between NT, Unix and MVS



Experience Using Statistical Usage Testing at the FAA

Ara Kouchakdjian
Ara.Kouchakdjian@q-labs.com
+1 301 864 0359

Q-Labs Group
Sweden, Germany, Norway, U.S.A.

Agenda

- ◆ The Project
- ◆ Statistical Usage Testing
- ◆ Timeline
- ◆ Technology Acquisition
- ◆ Usage Model Development
- ◆ Test Generation and Execution
- ◆ Lessons Learned
- ◆ Broader Insights
- ◆ Next Steps

The Project

- ◆ Support System for Air Traffic Controllers
 - Information system to support controllers in making longer terms decisions
 - Not a control system
 - >>300 KSLOC
- ◆ FAA had responsibility for test/acceptance and subsequent fielding of this as a part of a larger system
- ◆ Decided to use a usage-based testing approach
 - Focus on field reliability
 - Typical 'acceptance' test felt to be insufficient

Statistical Usage Testing

- ◆ Test the software the way users use it
 - Develop test without code knowledge
 - Provide maximum field reliability gain
 - Provide user focus for quality
- ◆ Testing as a statistical experiment
 - Statistical inference
 - Testing decisions based on objective data

Why SUT?

- ◆ To maximize field reliability
- ◆ To maximize the potential for objective measurement
 - Measurement for test planning
 - Measurement for test execution
 - Measurement for product certification
 - Measurement for process assessment
- ◆ To remove human bias from testing
- ◆ To test consistently with field usage
- ◆ To make software quality assessment more scientific

SUT Process

- Test Planning
 - Define Test Goals / Stratify
 - Define Boundary and Stimuli
 - Define a Use
 - Allocate Test Budget to Goals
 - Allocate Test Approaches to Goals
- Usage Model Development
 - Build Model Structure
 - Assign Probabilities
 - Analyze Model
- Generate Tests
 - Generate Scripts
 - Process into Test Cases
 - Study Coverage
 - Compute Expected Results
- Test Execution
 - Run Tests
 - Determine Success/Failure
 - Record Pass/Fail
- Certification
 - Compute Quality Measures
 - Compute Sppage Criteria
 - Make Release Decisions

Timeline

- ◆ Initial Training: Summer 1997
- ◆ Initial Modeling: Summer 1997-Summer 1998
- ◆ Testing: Spring 1998-Present
- ◆ Additional Training: Spring 1998
- ◆ Extensions/Modifications to Models: Summer 1998 - Present
- ◆ Additional Systems to Model: Summer 1998 - Present

Technology Acquisition

- ◆ Initial training - 3-4 day course to FAA staff
- ◆ ToolSET_Certify® delivered to FAA
- ◆ Q-Labs did initial modeling
- ◆ FAA focused on testing
- ◆ FAA took on modeling responsibilities with minor amounts of coaching

Usage Model Development

- ◆ 7 major models (state transition diagrams)
 - Corresponded to major menu pull-downs
 - Composed from 50+ smaller models
 - Major models had >350 states
- ◆ Models were at a mouse click level of granularity (arcs values)
- ◆ Represented all possible sequences of inputs
- ◆ Customer defined probabilities on arcs exiting each state

Test Generation and Execution

- ◆ Approximately 20 tests automatically generated per model
 - All executed, failures recorded
 - Looked at arc and state coverage based on sample
- ◆ Generated and executed additional tests for models with low coverage
- ◆ Sufficiency of test based on state and arc coverage
- ◆ Failures recorded by type, severity and where in test they occurred

Lessons Learned (1)

- ◆ Test cases appeared sufficient
 - Very few other tests run
 - Approach works for this type of software
- ◆ Test quite efficiently executed
 - Tests on one screen
 - Target software on another
 - Software easy for testers to directly validate
- ◆ Elimination of paper
 - Tests viewed/executed electronically
 - Infinite numbers generated at any time
 - No need for paper test instructions

Lessons Learned (2)

- ◆ Testing effective
 - Non-modeled components were more reliable prior to FAA testing (used SUT on the 'worst' subsystem)
 - Modeled components more reliable after FAA test
- ◆ Found large numbers of requirements and 'what if' issues
 - Result of usage models being precise descriptions of behavior
 - Allowed questions to be concrete

Broader Insights

- ◆ Model based approach eliminated human bias
 - Tested what one would not have otherwise tested
- ◆ Can test 'forever'
 - Always generate 'fresh' tests
- ◆ Change in testing paradigm
 - Go away from waiting until end to think about test and creating detailed procedures
 - Shift to initial investment which pays dividends throughout
- ◆ Large models, relates to complexity of functionality

Next Steps

- ◆ Have modeled another subsystem
- ◆ Will model remainder of system

- ◆ Convinced that this is the way to test
 - Cannot imagine writing test procedures
 - Continue to use model generated tests to this day

- ◆ Gathering up quantitative results (currently doing installations)