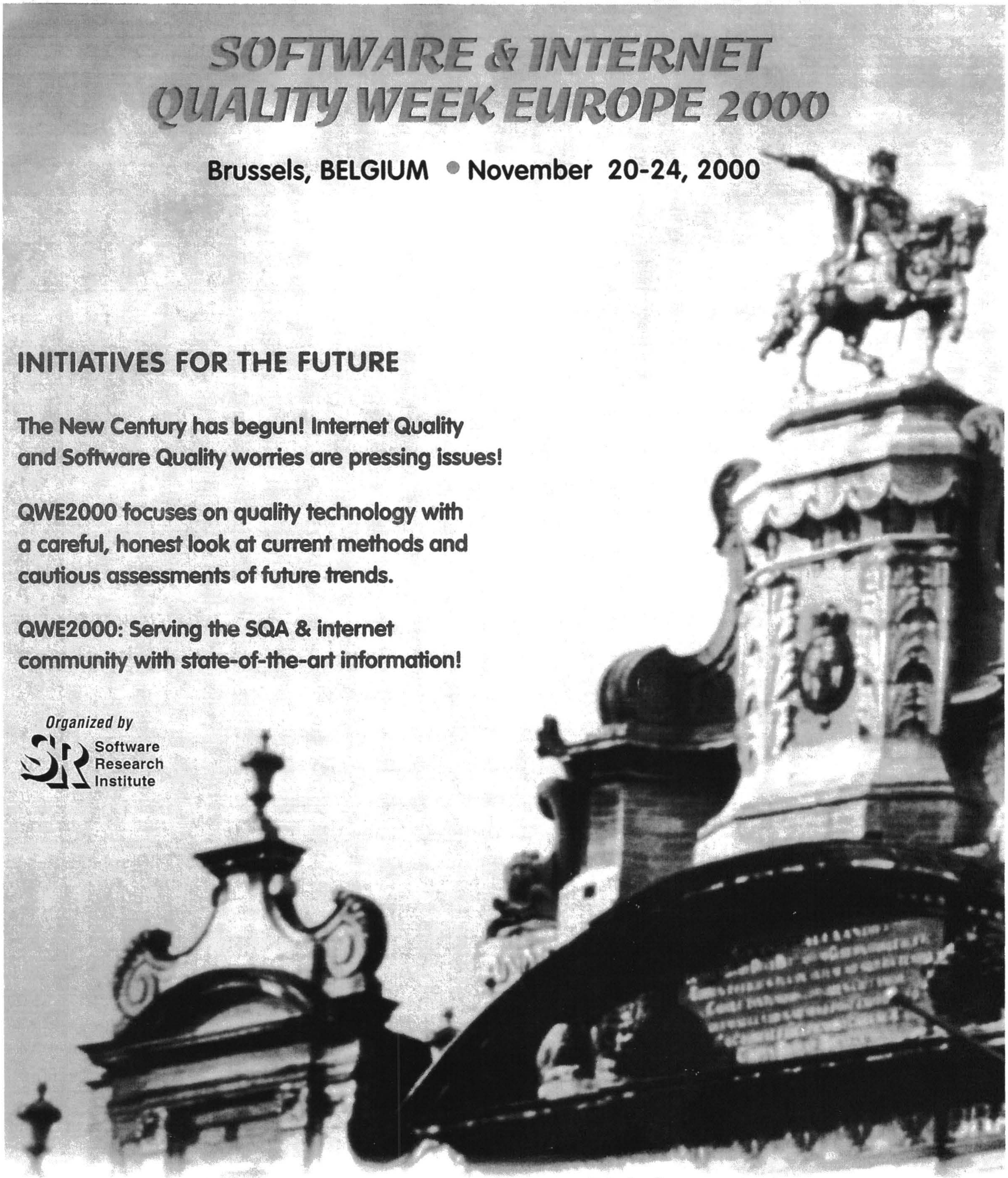
SOFTWARE & INTERNET QUALITY WEEK EUROPE 2000

Brussels, BELGIUM • November 20-24, 2000



INITIATIVES FOR THE FUTURE

The New Century has begun! Internet Quality and Software Quality worries are pressing issues!

QWE2000 focuses on quality technology with a careful, honest look at current methods and cautious assessments of future trends.

QWE2000: Serving the SQA & internet community with state-of-the-art information!

Organized by

Software Research Institute

In cooperation with

American Society for Quality ASQ Software Division

ESI European Software Institute

TestNet

SAI

European ESSI Systems & Software Institute

Industry Sponsors

CMG Information Technology

elementool

eValid

gitek interaction through software

ps_testware Software Testing Services

TESCOM

QWE 2000 Speaker Evaluation
A.M. Monday, 20 November, 2000

Tutorial: □A1 □B1 □C1 □D1

Speaker: ______________________________

Overall rating of presentation

(5 = excellent, 1 = poor):

5	4	3	2	1
□	□	□	□	□

Comments: ______________________________

P.M. Monday, 20 November, 2000

Tutorial: □A2 □B2 □C2 □D2

Speaker: ______________________________

Overall rating of presentation

(5 = excellent, 1 = poor):

5	4	3	2	1
□	□	□	□	□

Comments: ______________________________

Please return this form to the registration desk by 10:30 AM, Friday 24 November, at the very latest! Thank you!

QWE 2000 Speaker Evaluation
A.M. Tuesday, 21 November, 2000

Tutorial: □E1 □F1 □G1 □H1

Speaker:______________________________

Overall rating of presentation

(5 = excellent, 1 = poor):

5	4	3	2	1
□	□	□	□	□

Comments:______________________________

____

____

____

____

____

____

____

____

P.M. Tuesday, 21 November, 2000

Tutorial: □E2 □F2 □G2 □H2

Speaker:______________________________

Overall rating of presentation

(5 = excellent, 1 = poor):

5	4	3	2	1
□	□	□	□	□

Comments:______________________________

____

____

____

____

____

____

____

____

Please return this form to the registration desk by 10:30 AM, Friday 24 November, at the very latest! Thank you!

TUTORIAL PRESENTATIONS

Fourth International
Software & Internet
Quality Week Europe 2000

20-24 November 2000

The Sheraton Brussels
3 Place Rogier
B-1210 Brussels, Belgium

Software Research Institute
901 Minnesota Street
San Francisco, CA 94107 USA

Phone: +1 (415) 550-3020 – FAX: +1 (415) 550-3030

PRE-CONFERENCE TUTORIALS

Monday, 20 November, 2000

Morning Classes: 8:30 - 10:00 & 10:30- 12:00
Refreshment Break: 10:00-10:30

A1 Web Testing Masterclass - Part 1

Dr. Gualtierio Bazzana, ONION, S.P.A., Italy

- The tutorial focuses on testing methods and tools which can be successfully applied to the testing of Web-based applications, notably, presented from a technical point of view
- Internet WWW servers
- Intranet dynamic applications
- Extranet e-commerce application

B1 Risk Based Test Effort Estimation with Test Point Analysis

Mr. Ruud Teunissen and Rob Baarda, Gitek, Belgium

- Well defined steps from business risks to test coverage
- Early and stronger test involvement of all parties concerned
- Useful for all tests

C1 Component Architecture and Business Models

Dr. Alan Cameron Wills, TriReme International Ltd.

- Flexible systems are built from pluggable components
- Pluggable components need interoperable architecture
- Interoperability requires unambiguous business models

D1 ISEB Software Testing Foundation Certificate - Part 1

Mike Russell, Insight Consulting Ltd., in association with Systeme Evolutif

The Information Systems Examination Board (ISEB) of the British Computer Society has accredited the standard full course for delegates of Quality Week Europe, with some experience in testing, who wish to take the examination leading to the Foundation Certificate in Software Testing. The exam will be offered during the conference, supervised by an ISEB invigilator.

TUTORIAL DAY LUNCH & NETWORKING: 12:00 - 13:30
Afternoon Classes: 13:30-15:00 & 15:30-17:00
Refreshment Break: 15:00-15:30

A2 Web Testing Masterclass - Part 2

Dr. Gualtierio Bazzana, ONION, S.P.A, Italy

This tutorial will deal with testing management aspects which are fundamentally affected by the nature of Web applications, including: RAD, regression issues. Testing solution will then be presented, both for static aspects (related to HTML, pictures, XML) and dynamic aspects (ASP, CGI, Proxies, Cookies, etc.)

B2 A Quantitative Risk Assessment Model For Software Quality, Testing and Safety

Ms. Alice Lee and Dr. Eric Wong, NASA Johnson Space Center and Telcordia Technologies, USA

- Quantitative risk management for software
- Test efficiency and improvement
- Software quality and maturity level improvement

C2 On A Standards Based Quality Framework for Web Portals

Dr. Hans-Ludwig Hausen, GMD, Germany

- Are web portals so peculiar that the advantages of defining a quality model for web portal over comes the disadvantage of having standard proliferation?
- How can the quality characteristics/sub-characteristics defined and suggested by such attributes as given by, for example, ISO 9126 be covered by the expected properties of a web portal?
- What are possible quality profiles (i. e., lists of expected relative "values" of characteristics/ sub-characteristics) for web portals according to a model conform to a quality standard, e.g. ISO 9126, ISO12119, ISO14598, QoS?

D2 ISEB Software Testing Foundation Certificate - Part 2

Mike Russell, Insight Consulting Ltd., in association with Systeme Evolutif

Syllabus Topics for the Foundation Certificate in Software Testing: Principles of Testing, Testing thoughout the lifecycle, Dynamic Testing Techniques, Static Testing, Test Management, and Tool Support for Testing (CAST).

Evening Class: 18:00-21:00

ISEB Software Testing Foundation Certificate - Part 3

Mike Russell, Insight Consulting Ltd., in association with Systeme Evolutif

Tuesday, 21 November, 2000

Morning Classes: 8:30 - 10:00 & 10:30- 12:00
Refreshment Break: 10:00-10:30

E1 Requirements Engineering for SW Developers and Testers

Tom Gilb, Result Planning Limited, Norway

System Requirements Engineering Course will teach an innovative fresh approach to system requirements. It is distinguished from all previous requirements approaches by its level of quantification of critical system requirements. Its all based on a practical, defined language for specification which produces rigor and clarity to any requirements specification process. All cases of system design requirements are encompassed, including quality requirements, resource requirements and design constraints.

F1 The Effective SQA Manager: Getting Things Done

Robert A. Sabourin, Purkinje Inc., Canada

This interactive tutorial walks you through several "down to earth" practical aspects of running an SQA team. The tutorial is presented in parable form. In this tutorial the audience will experience the real life problems encountered by a NOGO.COMs neophyte SQA Manager "Fred". "Fred" must turn around an enthusiastic but severely under staffed and under budget team of SQA professionals working in a chaotic development environment into a productive effective team! "Fred" is under the gun - he has to get things done!

G1 Cool Q-Quality Improvement for Multi-Disciplinary Tasks In Website Development

Adrian Cowderoy, Nexus World.net Limited, UK

The tutorial is targeted at website producer/directors and managers. It is also strongly recommended for software quality people who are moving into the Internet business. The tutorial addresses website content and structure, and the functionality resulting from using Internet development tools.

H1 ISEB Software Testing Foundation Certificate - Part 4

Mike Russell, Insight Consulting Ltd., in association with Systeme Evolutif

TUTORIAL DAY LUNCH & NETWORKING: 12:00 - 13:30
Afternoon Classes: 13:30-15:00 & 15:30-17:00
Refreshment Break: 15:00-15:30

E2 Specification Quality Control (SQC): A Practical Inspection Workshop on Requirements Specification

Tom Gilb, Result Planning Limited, Norway

40-60% of all software bugs which escape test to the field user have been traced to requirements and design specifications before coding. It has then been proved that Inspecting the specifications sharply reduces this problem. This workshop will explore all aspects of Specification Quality Control in a hands on practical workshop. Participants will actively experience the technology necessary to attack this quality challenge.

F2 The Quality Challenge for Network-Based Software Systems

Tom Drake, Integrated Computer Concepts, Inc., USA

- Internet quality for network centric systems
- Enterprise testing
- Quality network systems theory
  (See elaborate abstract on web page)

G2 WebSite Testing

Tobias Mayer and Dr. Edward Miller, eValid, Inc., USA

- In the past few years WebSites have grown from simple, static collections of HTML pages to complex pieces of software using advanced technologies including ASP, XML, script languages, e-commerce and more
- Testing of such complexity is a forever-growing challenge
- Testing of passive vs. interactive pages & static vs. dynamic pages will be explored
- The use of a 'Test Enabled Web Browser' will be emphasized as the most effective way of realistically testing most WebSites

H2 ISEB Software Testing Foundation Certificate - Part 5

Mike Russell, Insight Consulting Ltd., in association with Systeme Evolutif

ISEB Software Testing Foundation Certificate - Exam

Note: The exam will take place on Wednesday morning at 8:30 am. Certificates will be granted during the awards ceremony to be announced at the conference.

17:30 - 19:00 CONFERENCE WELCOME RECEPTION

QWE2000 Tutorial D1

Mike Russell
(Insight Consulting Ltd., Ireland
in association with Systeme Evolutif, UK)

"ISEB Software Testing Foundation Certificate"

The Information Systems Examination Board (ISEB) of the British Computer Society has accredited the standard full course for delegates of Quality Week Europe, with some experience in testing, who wish to take the examination leading to the Foundation Certificate in Software Testing. The course will take 18 hours, including the 'closed book' exam; which consist of forty multiple-choice questions.

The exam will be offered during the conference, supervised by an ISEB invigilator. The results and certificates will be presented at a special announcement during QWE2000.

Objectives of the Qualifications

Through the Software Testing qualifications offered by ISEB, the Subject Board has the following objectives:

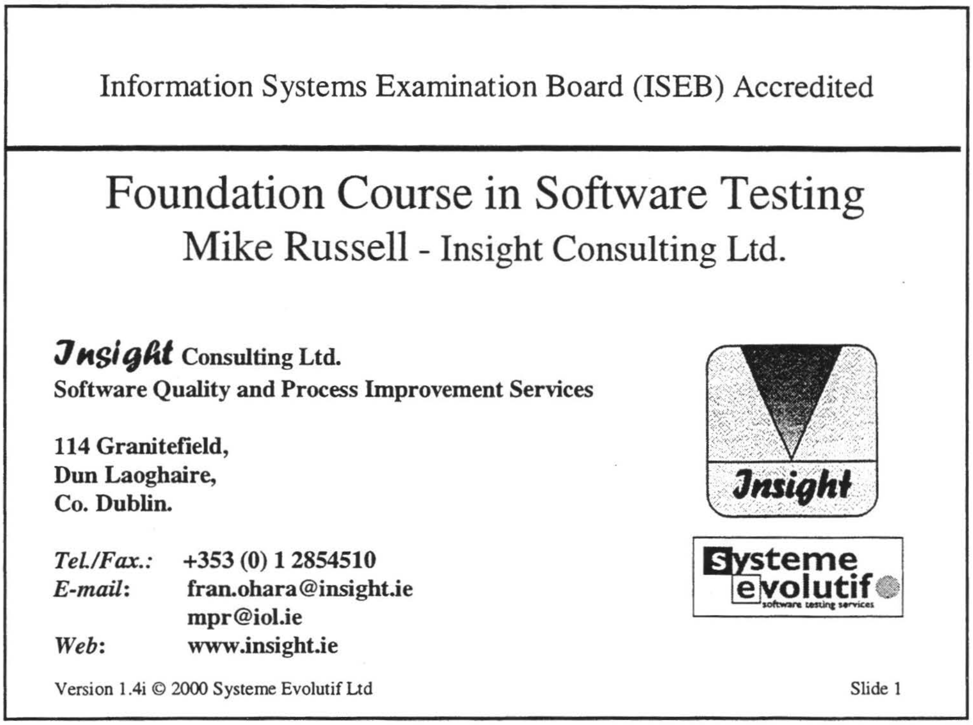
- To gain industry recognition for testing as an essential and professional software engineering specialization.
- Through the BCS Professional Development Scheme and the Industry Structure Model, provision of a standard framework for the development of testers' careers.
- To enable professionally qualified testers to be recognized by employers, customers and peers, and to raise the profile of testers.
- To promote consistent and good testing practice within all software engineering disciplines.
- To identify testing topics that are relevant and of value to industry.
- To enable software suppliers to hire certified testers and thereby gain commercial advantage over their competitors by advertising their tester recruitment policy.
- To provide an opportunity for testers or those with an interest in testing to acquire an industry recognized qualification in the subject.

Presentation Abstract

Syllabus Topics for the Foundation Certificate in Software Testing:

- Principles of Testing
- Testing thoughout the lifecycle
- Dynamic Testing Techniques
- Static Testing
- Test Management
- Tool Support for Testing (CAST).

Table of Contents



Information Systems Examination Board (ISEB) Accredited

Foundation Course in Software Testing

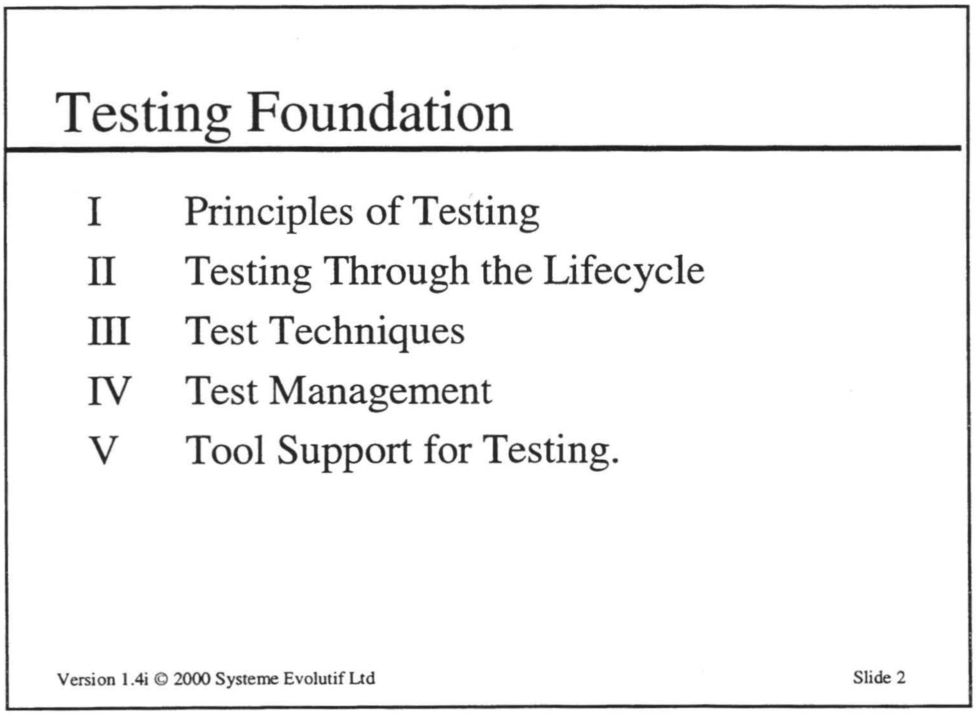
Mike Russell - Insight Consulting Ltd.

Insight Consulting Ltd.
Software Quality and Process Improvement Services

114 Granitefield,
Dun Laoghaire,
Co. Dublin.

Tel./Fax.: **+353 (0) 1 2854510**
E-mail: **fran.ohara@insight.ie**
mpr@iol.ie
Web: **www.insight.ie**

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 1



Testing Foundation

I Principles of Testing

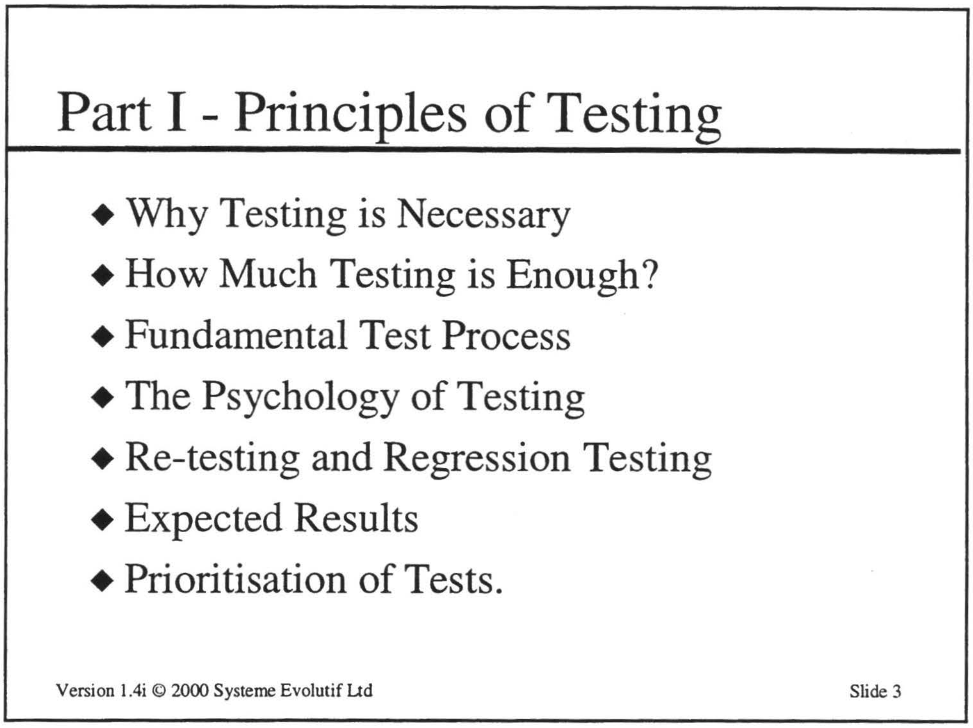
II Testing Through the Lifecycle

III Test Techniques

IV Test Management

V Tool Support for Testing.

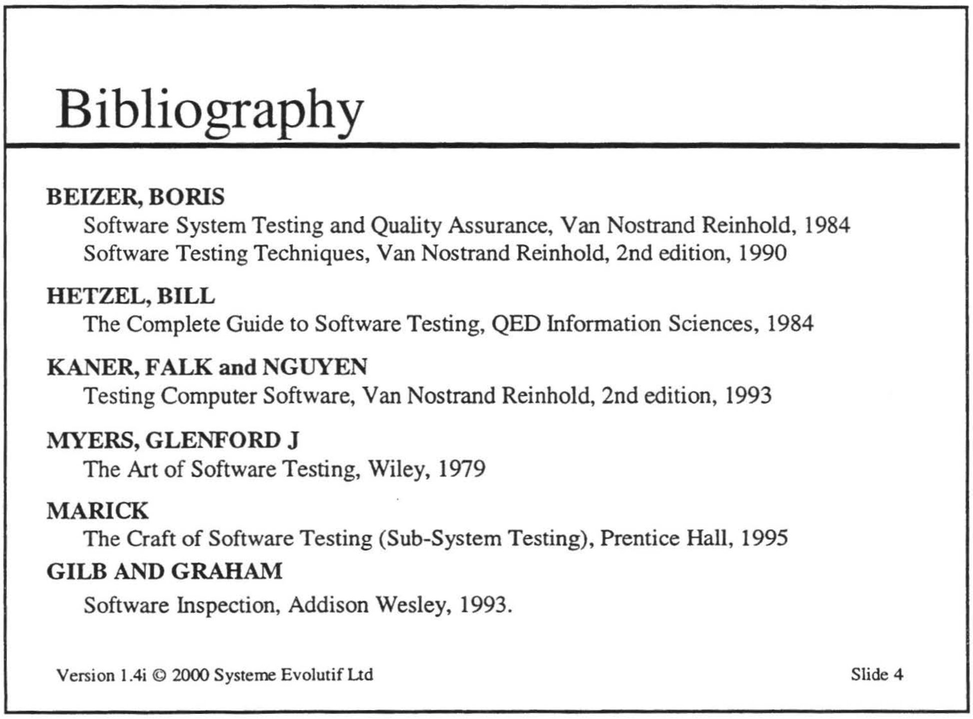
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 2



Part I - Principles of Testing

- ◆ Why Testing is Necessary
- ◆ How Much Testing is Enough?
- ◆ Fundamental Test Process
- ◆ The Psychology of Testing
- ◆ Re-testing and Regression Testing
- ◆ Expected Results
- ◆ Prioritisation of Tests.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 3



Bibliography

BEIZER, BORIS
Software System Testing and Quality Assurance, Van Nostrand Reinhold, 1984
Software Testing Techniques, Van Nostrand Reinhold, 2nd edition, 1990

HETZEL, BILL
The Complete Guide to Software Testing, QED Information Sciences, 1984

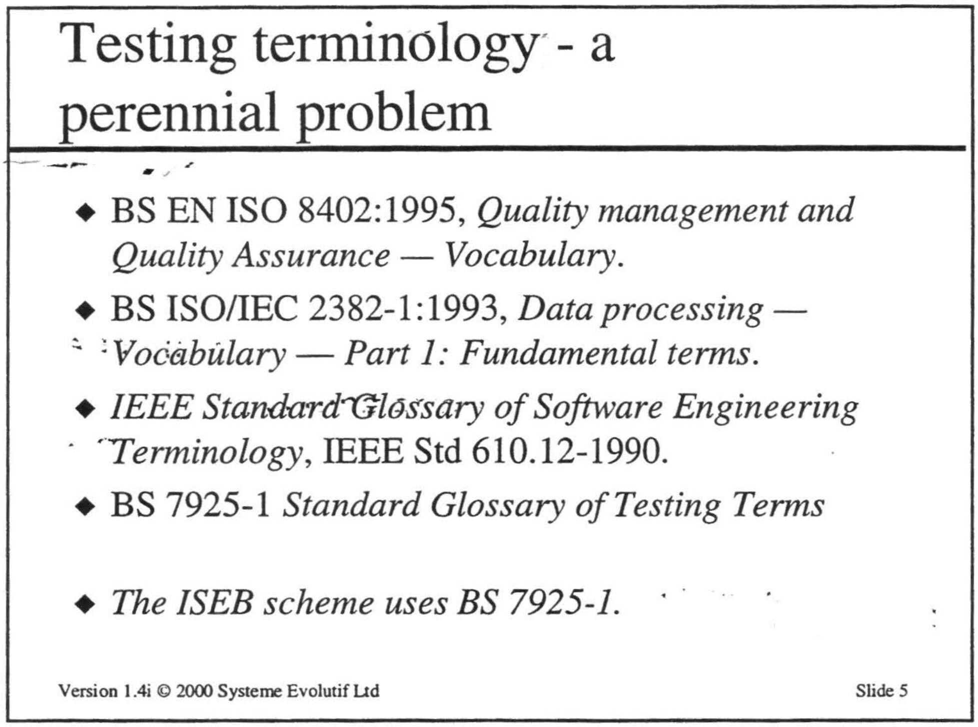
KANER, FALK and NGUYEN
Testing Computer Software, Van Nostrand Reinhold, 2nd edition, 1993

MYERS, GLENFORD J
The Art of Software Testing, Wiley, 1979

MARICK
The Craft of Software Testing (Sub-System Testing), Prentice Hall, 1995

GILB AND GRAHAM
Software Inspection, Addison Wesley, 1993.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 4

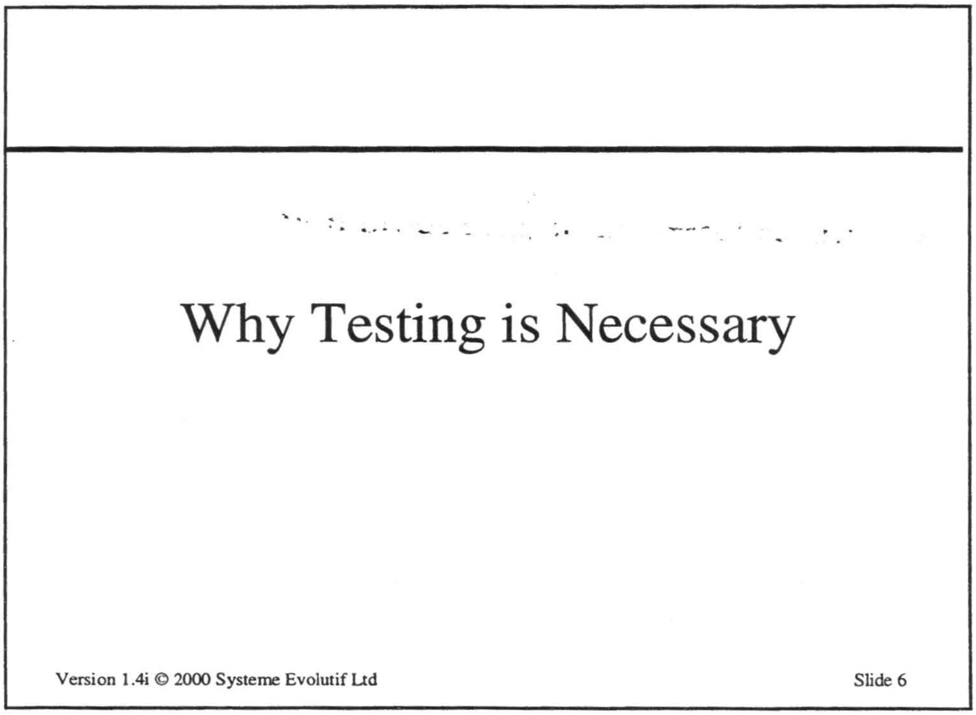


Testing terminology - a perennial problem

- BS EN ISO 8402:1995, *Quality management and Quality Assurance — Vocabulary.*
- BS ISO/IEC 2382-1:1993, *Data processing — Vocabulary — Part 1: Fundamental terms.*
- *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990.
- BS 7925-1 *Standard Glossary of Testing Terms*

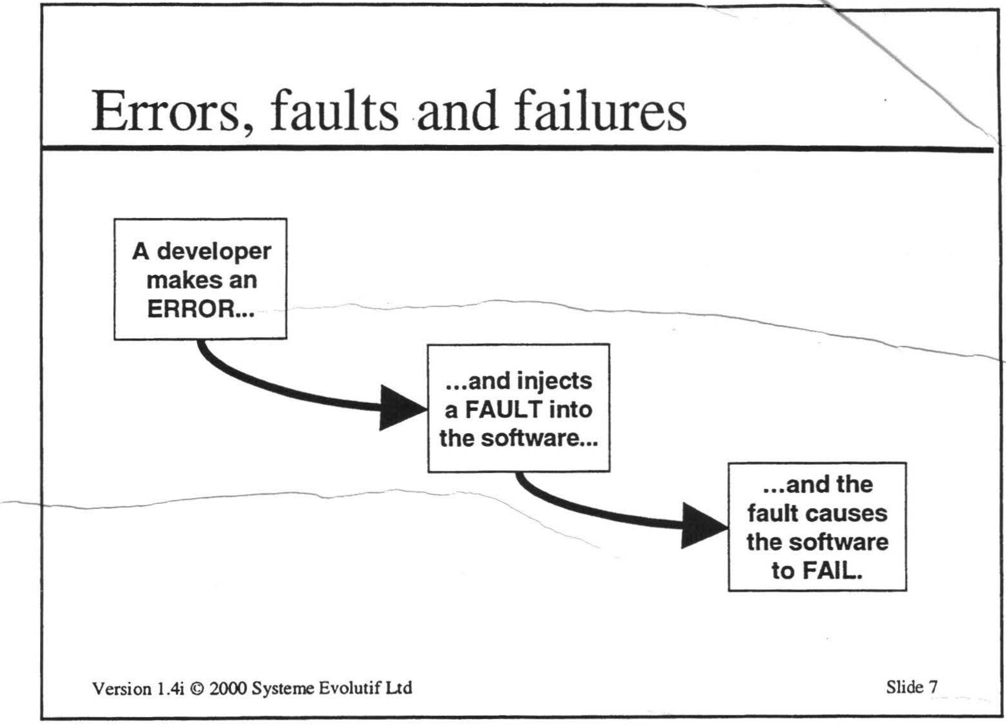
- *The ISEB scheme uses BS 7925-1.*

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 5



Why Testing is Necessary

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 6



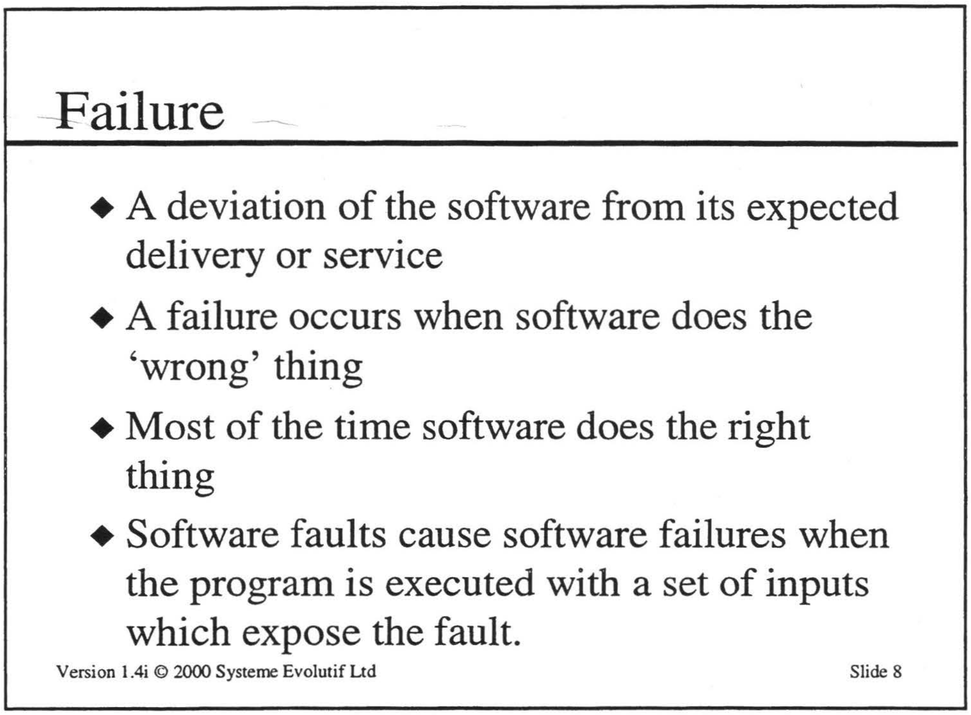
Errors, faults and failures

A developer makes an ERROR...

...and injects a FAULT into the software...

...and the fault causes the software to FAIL.

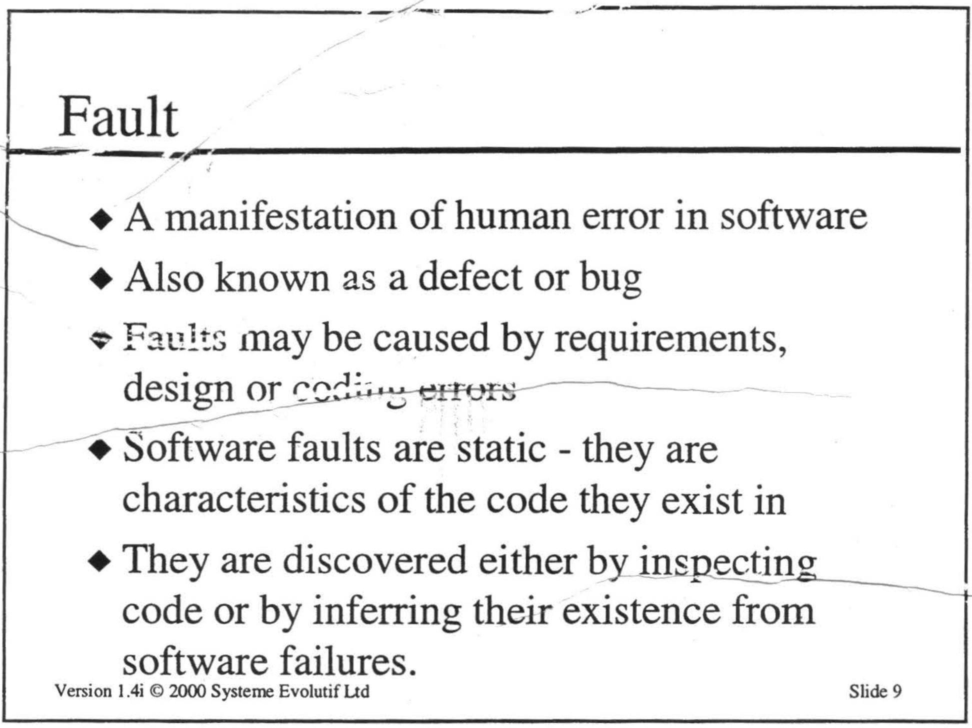
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 7



Failure

- A deviation of the software from its expected delivery or service
- A failure occurs when software does the ‘wrong’ thing
- Most of the time software does the right thing
- Software faults cause software failures when the program is executed with a set of inputs which expose the fault.

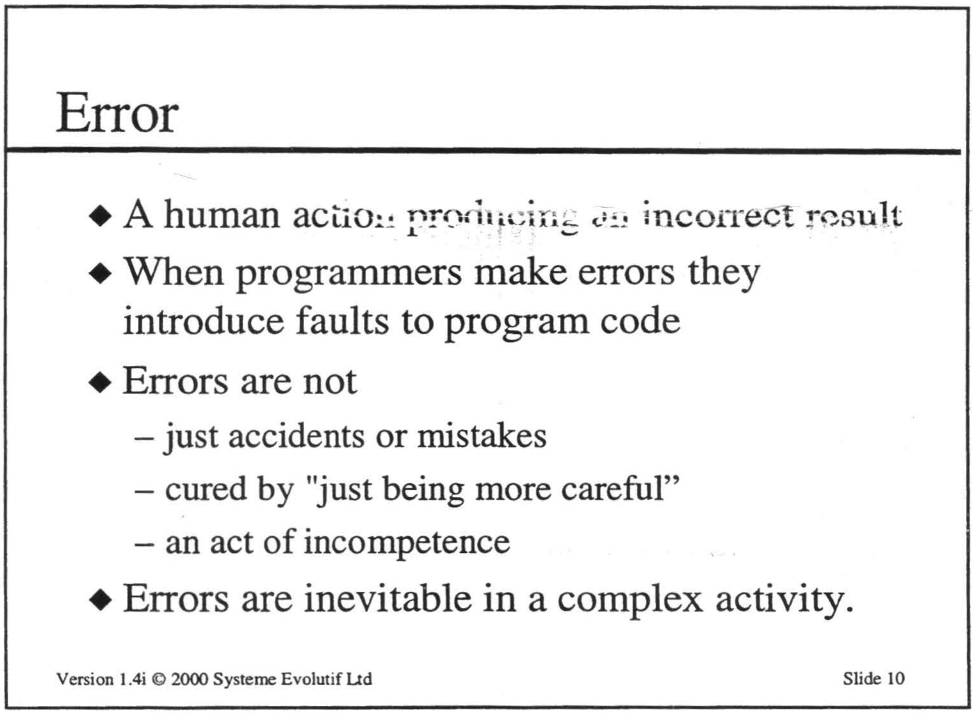
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 8



Fault

- A manifestation of human error in software
- Also known as a defect or bug
- Faults may be caused by requirements, design or coding errors
- Software faults are static - they are characteristics of the code they exist in
- They are discovered either by inspecting code or by inferring their existence from software failures.

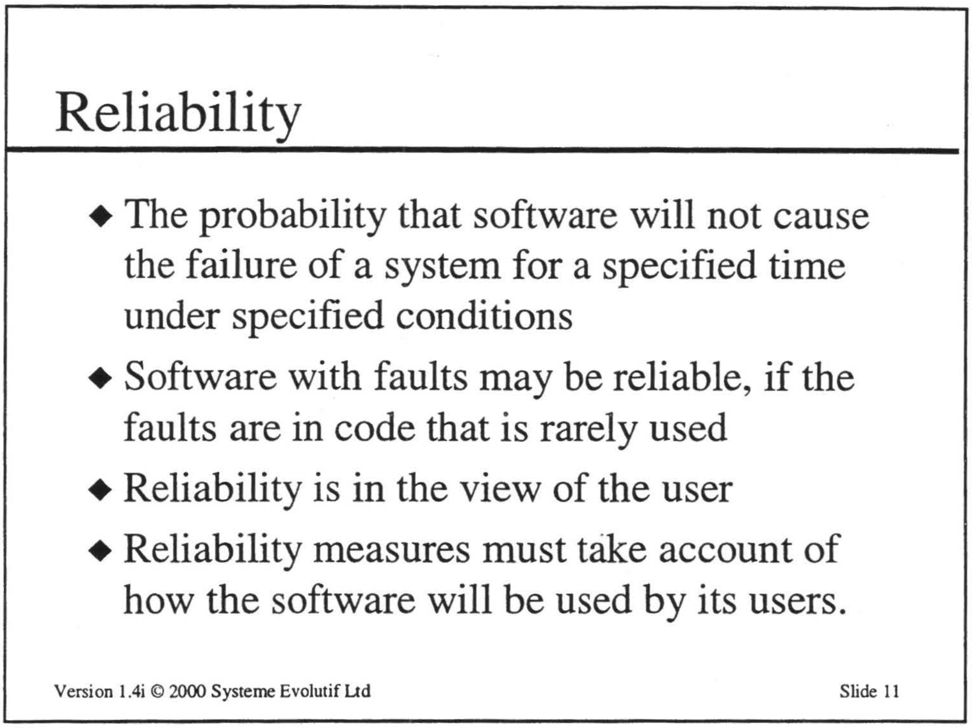
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 9



Error

- A human action producing an incorrect result
- When programmers make errors they introduce faults to program code
- Errors are not
  - just accidents or mistakes
  - cured by "just being more careful"
  - an act of incompetence
- Errors are inevitable in a complex activity.

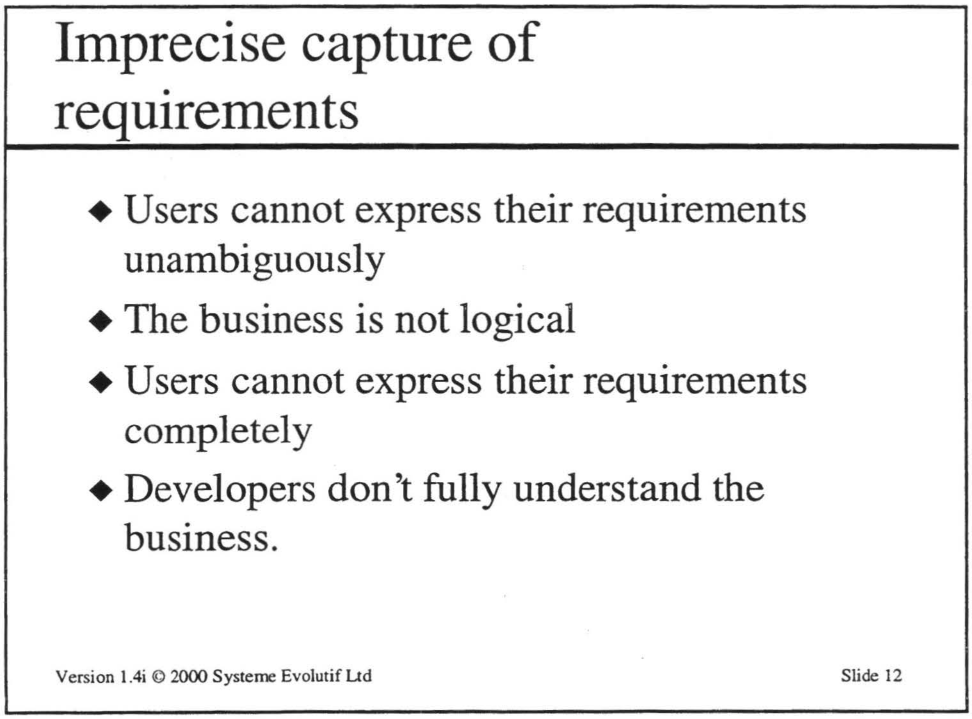
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 10



Reliability

- The probability that software will not cause the failure of a system for a specified time under specified conditions
- Software with faults may be reliable, if the faults are in code that is rarely used
- Reliability is in the view of the user
- Reliability measures must take account of how the software will be used by its users.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 11

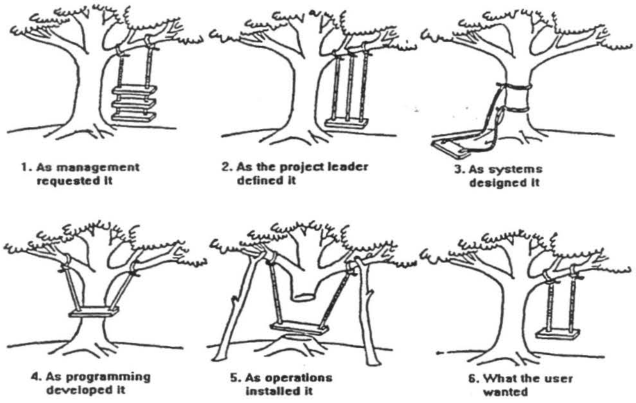


Imprecise capture of requirements

- Users cannot express their requirements unambiguously
- The business is not logical
- Users cannot express their requirements completely
- Developers don't fully understand the business.

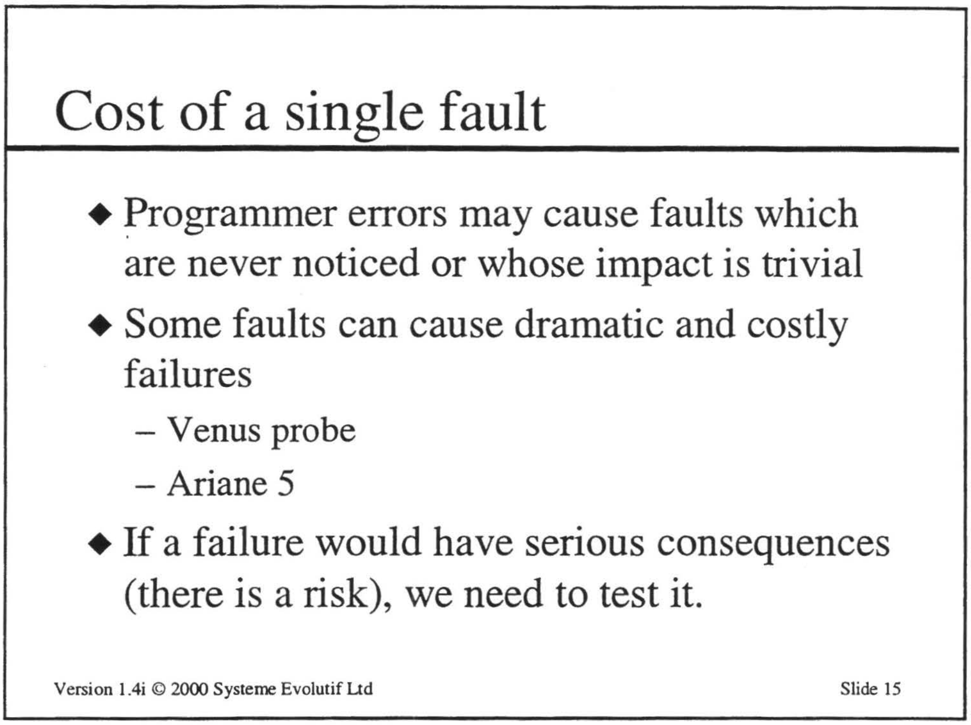
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 12

Progressive distortion


1. As management requested it
2. As the project leader defined it
3. As systems designed it
4. As programming developed it
5. As operations installed it
6. What the user wanted

Short term memory

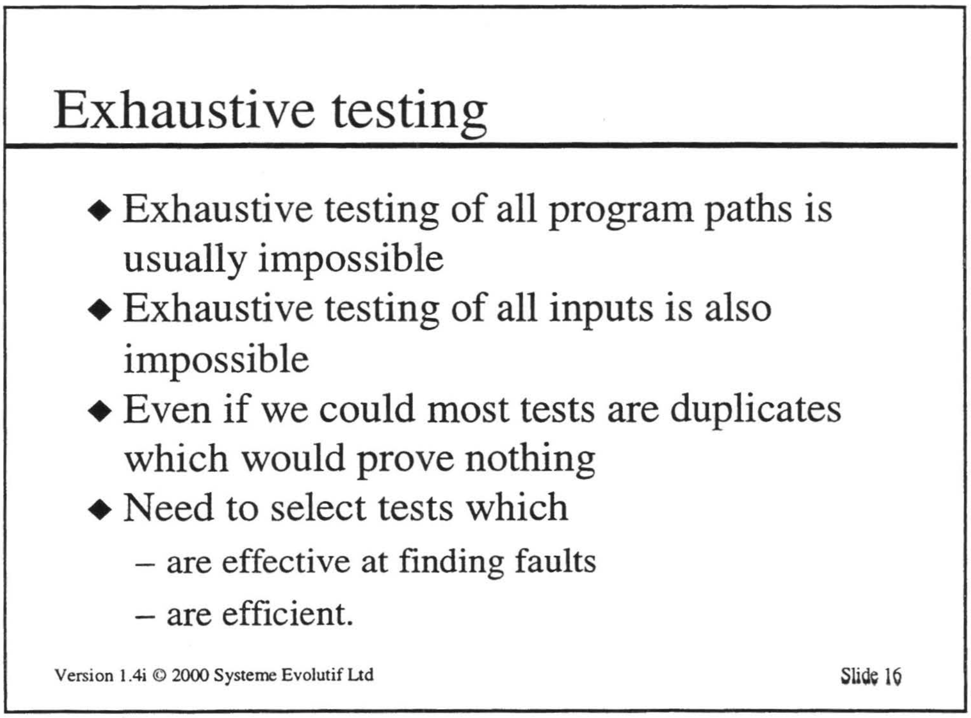
- Accounts for many detailed errors
  - we can only remember 5-9 immediately relevant details
  - short term memory overflows
  - e.g. we forget to define or initialise a variable.



Cost of a single fault

- Programmer errors may cause faults which are never noticed or whose impact is trivial
- Some faults can cause dramatic and costly failures
  - Venus probe
  - Ariane 5
- If a failure would have serious consequences (there is a risk), we need to test it.

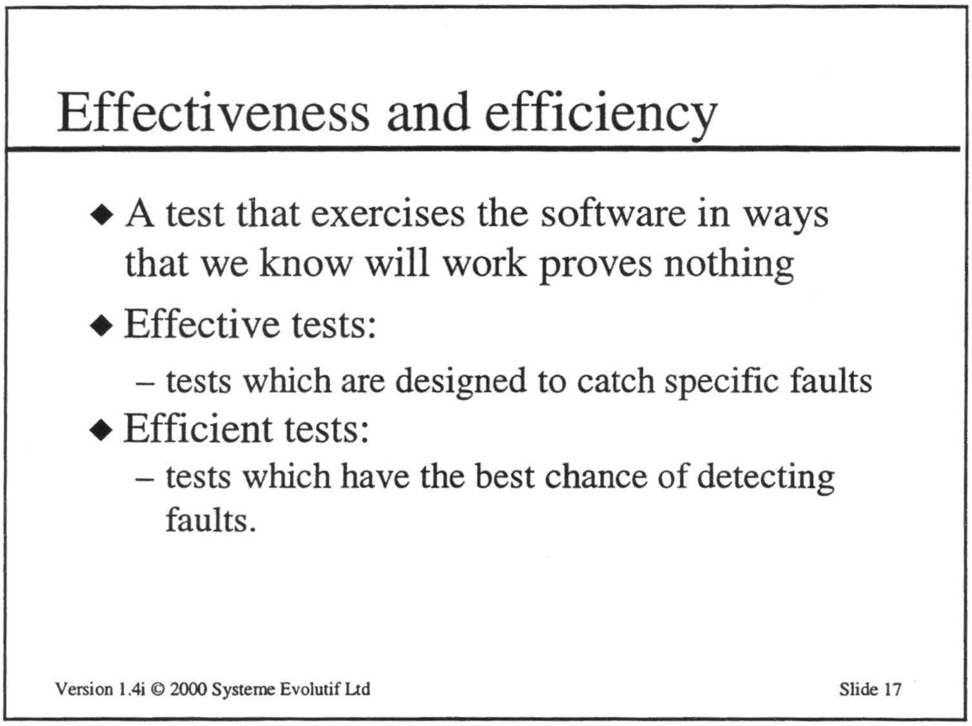
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 15



Exhaustive testing

- Exhaustive testing of all program paths is usually impossible
- Exhaustive testing of all inputs is also impossible
- Even if we could most tests are duplicates which would prove nothing
- Need to select tests which
  - are effective at finding faults
  - are efficient.

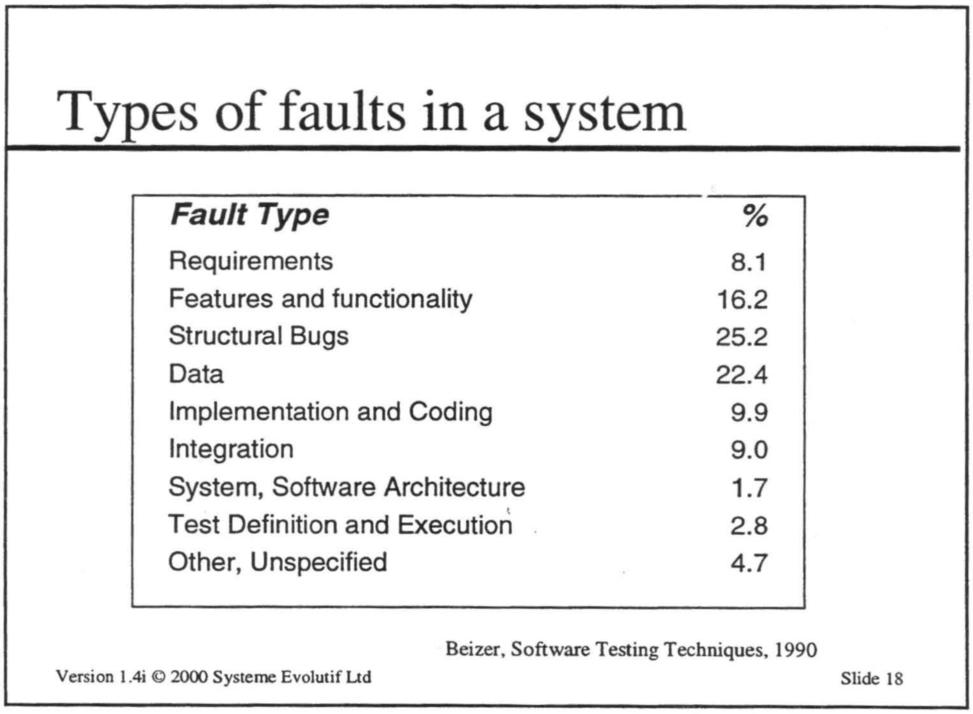
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 16



Effectiveness and efficiency

- A test that exercises the software in ways that we know will work proves nothing
- Effective tests:
  - tests which are designed to catch specific faults
- Efficient tests:
  - tests which have the best chance of detecting faults.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 17

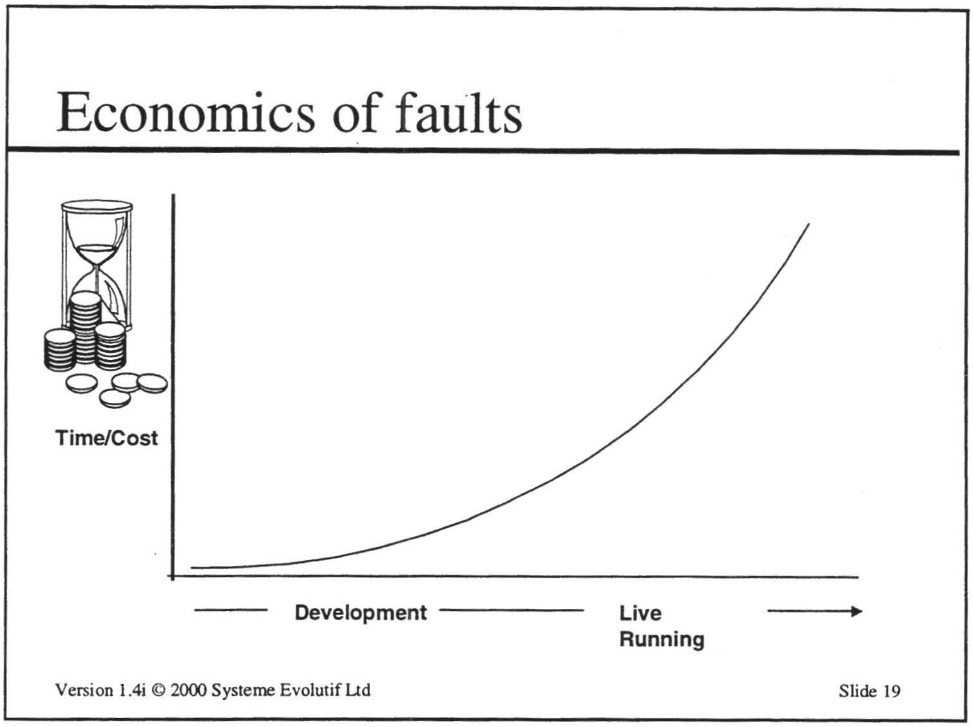


Types of faults in a system

Fault Type	**%**
Requirements	8.1
Features and functionality	16.2
Structural Bugs	25.2
Data	22.4
Implementation and Coding	9.9
Integration	9.0
System, Software Architecture	1.7
Test Definition and Execution	2.8
Other, Unspecified	4.7

Beizer, Software Testing Techniques, 1990

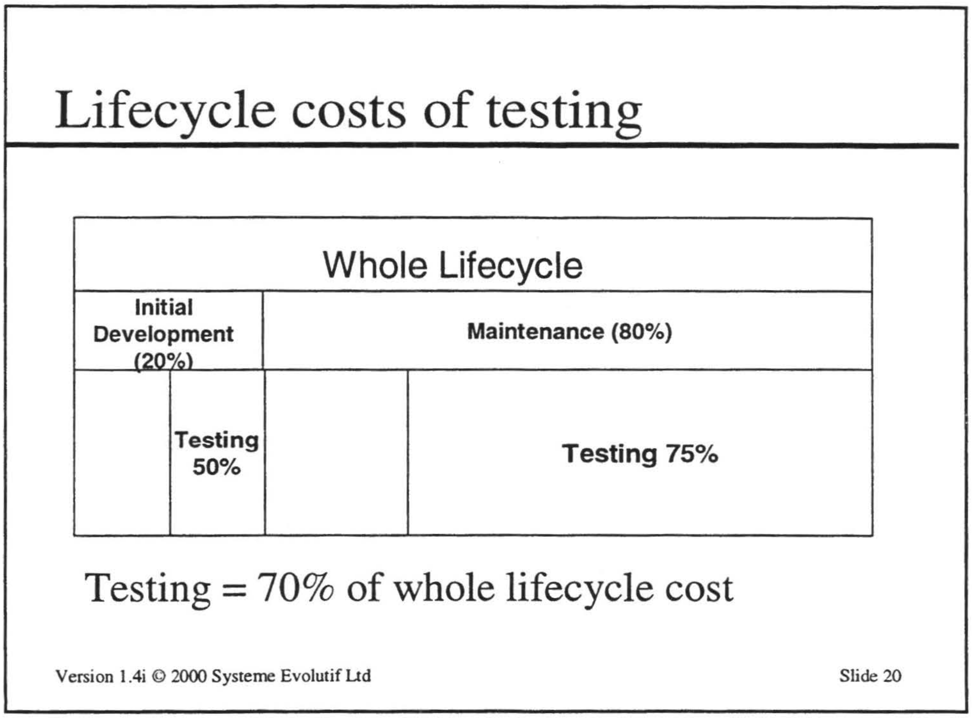
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 18



Economics of faults

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 19



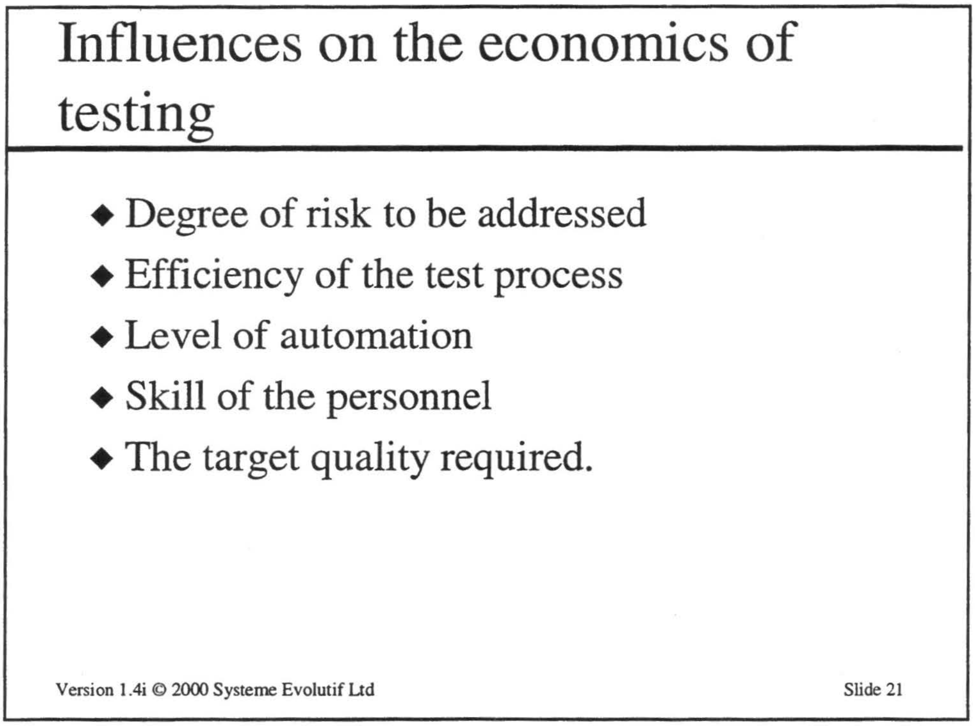
Lifecycle costs of testing

Whole Lifecycle			
Initial Development (20%)		Maintenance (80%)	
	Testing 50%		Testing 75%

Testing = 70% of whole lifecycle cost

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 20

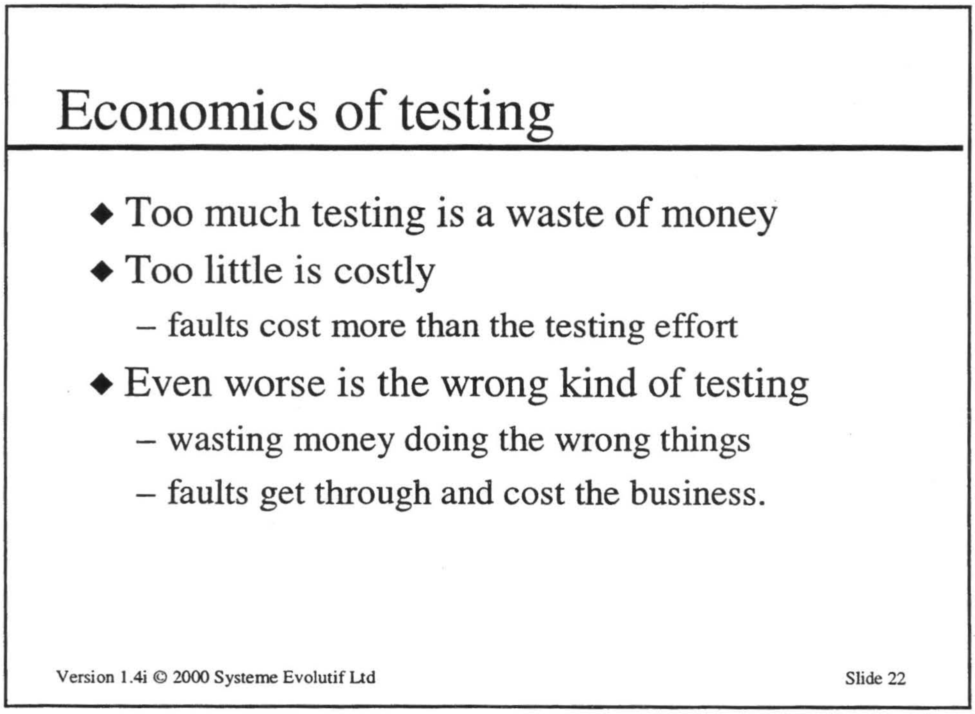


Influences on the economics of testing

- Degree of risk to be addressed
- Efficiency of the test process
- Level of automation
- Skill of the personnel
- The target quality required.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 21

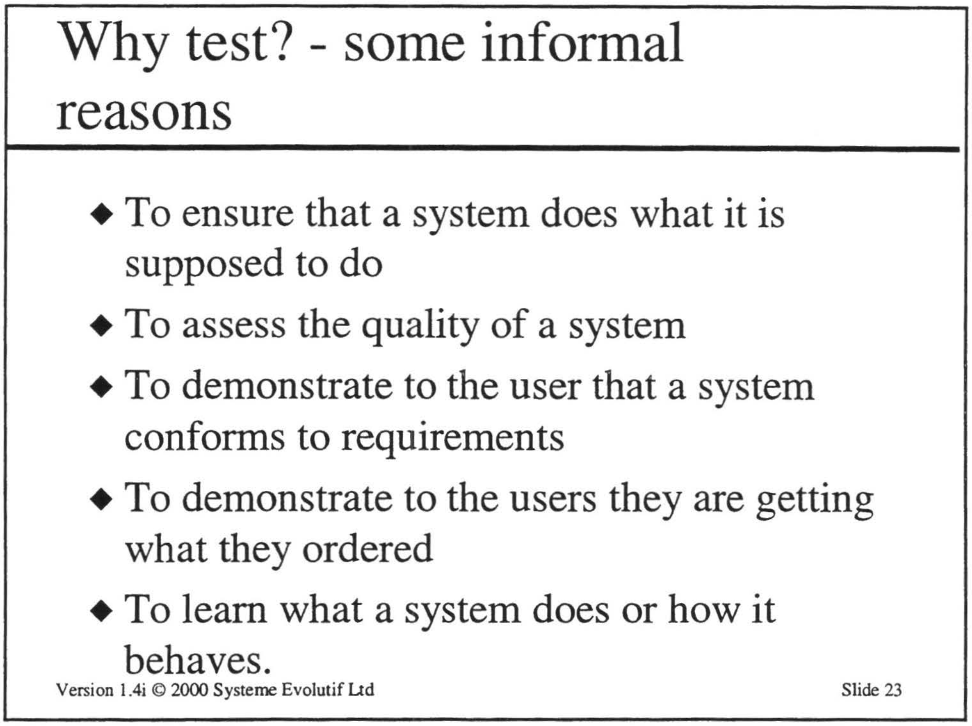


Economics of testing

- Too much testing is a waste of money
- Too little is costly
  - faults cost more than the testing effort
- Even worse is the wrong kind of testing
  - wasting money doing the wrong things
  - faults get through and cost the business.

Version 1.4i © 2000 Systeme Evolutif Ltd

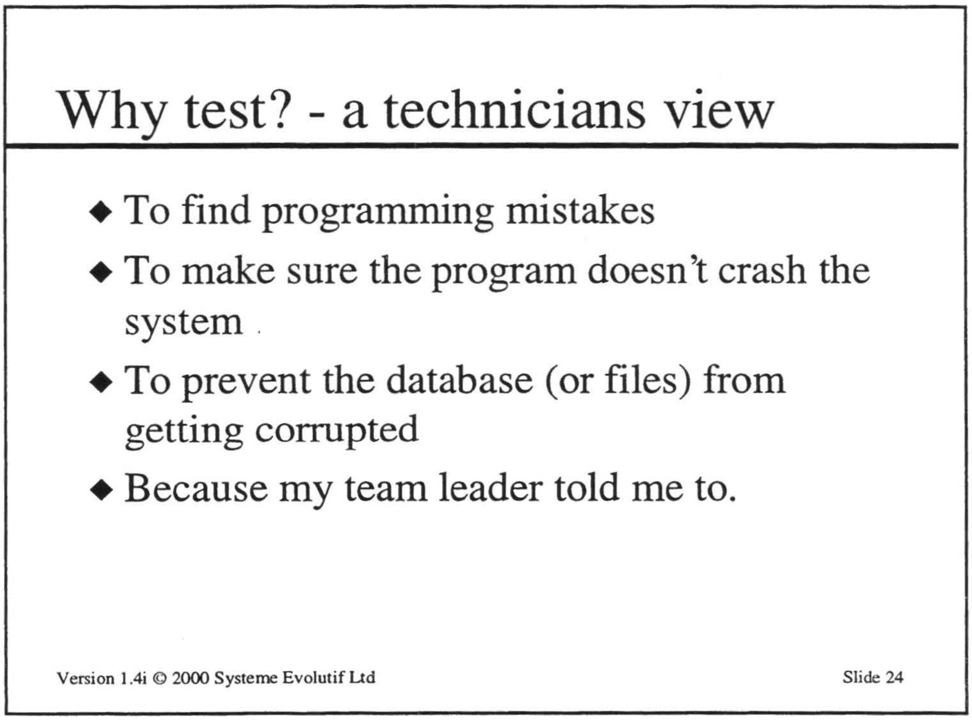
Slide 22



Why test? - some informal reasons

- To ensure that a system does what it is supposed to do
- To assess the quality of a system
- To demonstrate to the user that a system conforms to requirements
- To demonstrate to the users they are getting what they ordered
- To learn what a system does or how it behaves.

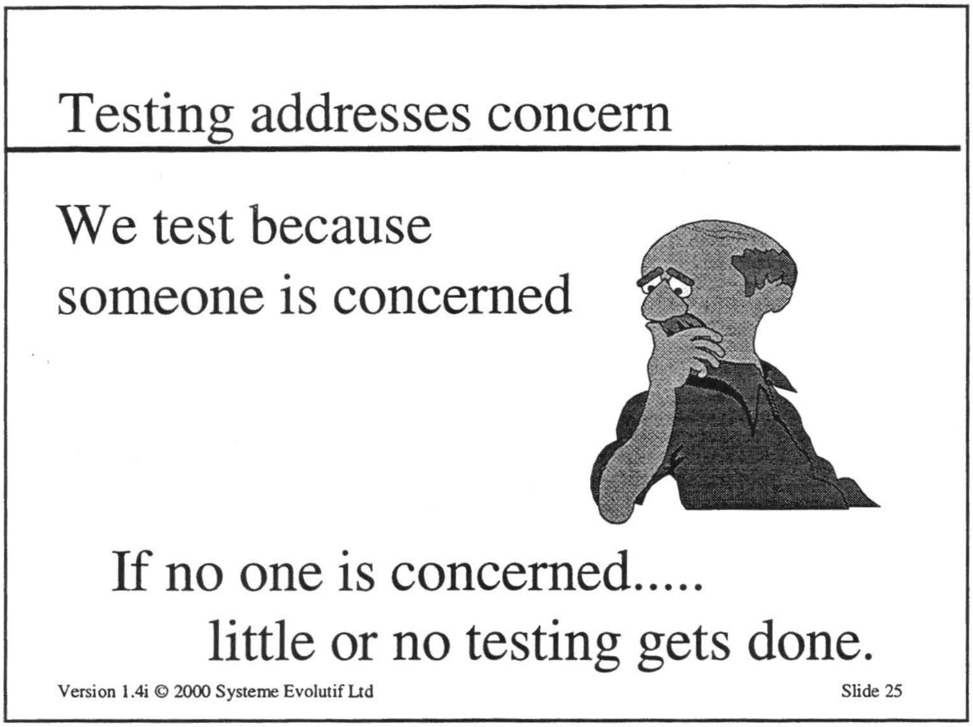
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 23



Why test? - a technicians view

- To find programming mistakes
- To make sure the program doesn't crash the system
- To prevent the database (or files) from getting corrupted
- Because my team leader told me to.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 24



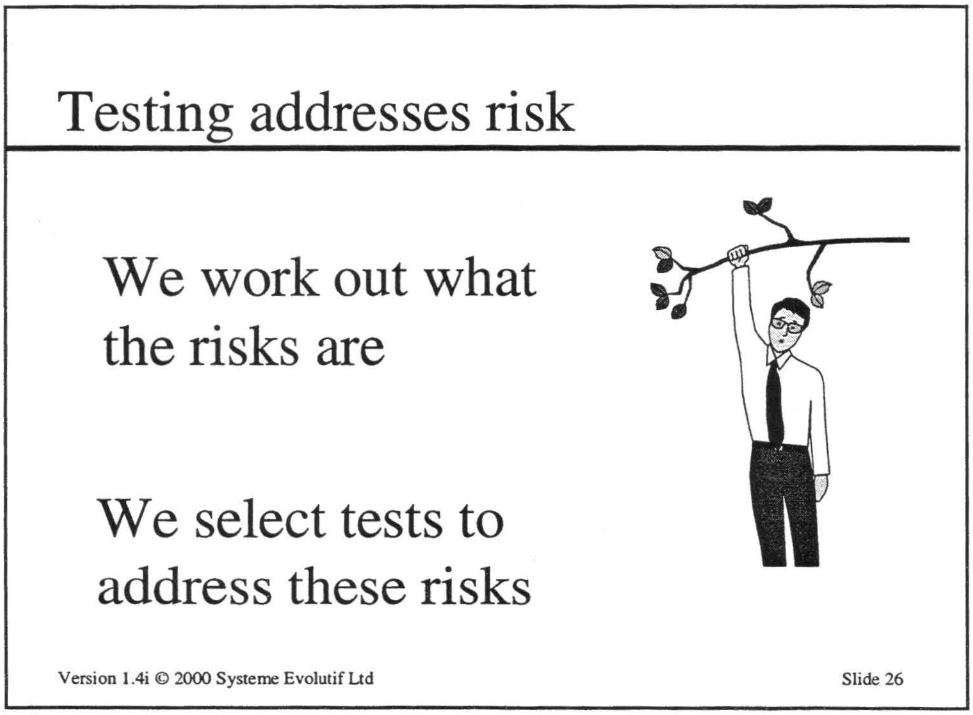
Testing addresses concern

We test because someone is concerned

If no one is concerned.....
little or no testing gets done.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 25



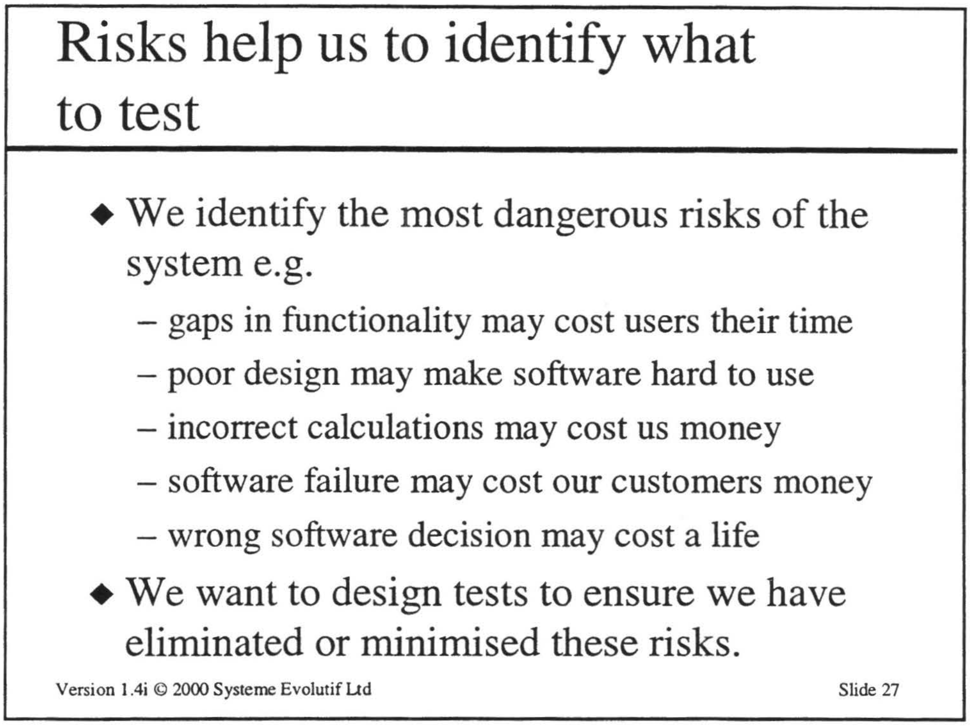
Testing addresses risk

We work out what the risks are

We select tests to address these risks

Version 1.4i © 2000 Systeme Evolutif Ltd

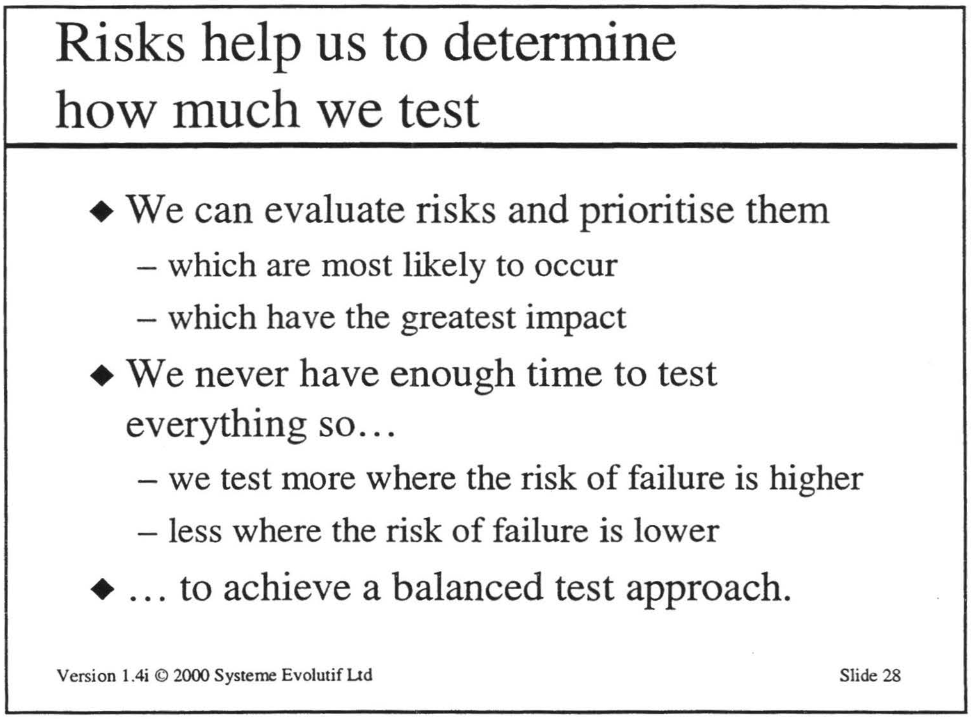
Slide 26



Risks help us to identify what to test

- We identify the most dangerous risks of the system e.g.
  - gaps in functionality may cost users their time
  - poor design may make software hard to use
  - incorrect calculations may cost us money
  - software failure may cost our customers money
  - wrong software decision may cost a life
- We want to design tests to ensure we have eliminated or minimised these risks.

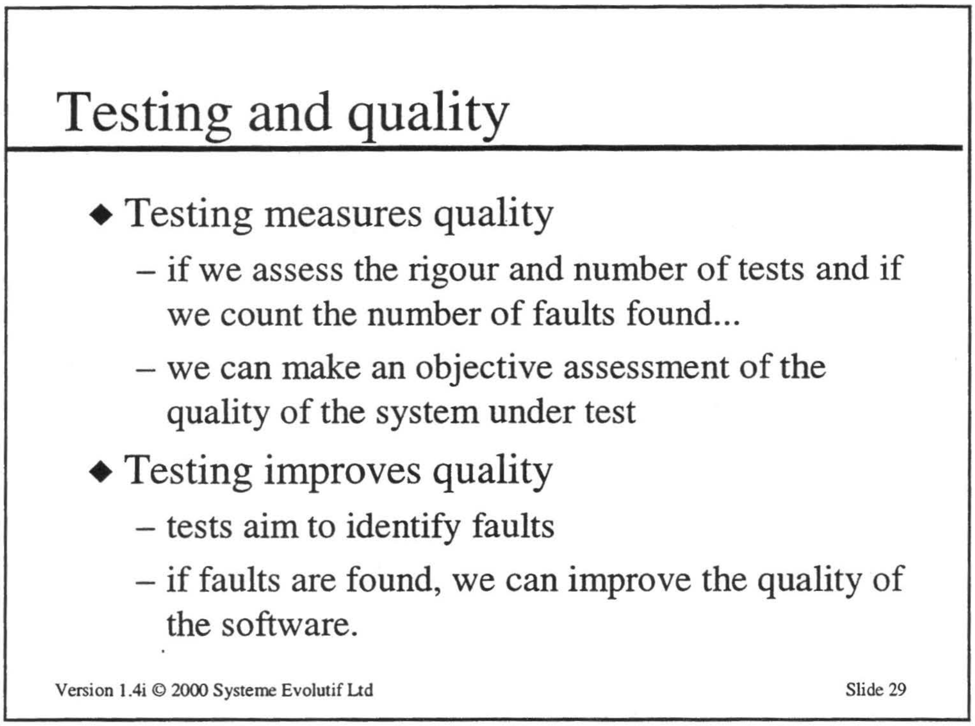
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 27



Risks help us to determine how much we test

- We can evaluate risks and prioritise them
  - which are most likely to occur
  - which have the greatest impact
- We never have enough time to test everything so…
  - we test more where the risk of failure is higher
  - less where the risk of failure is lower
- … to achieve a balanced test approach.

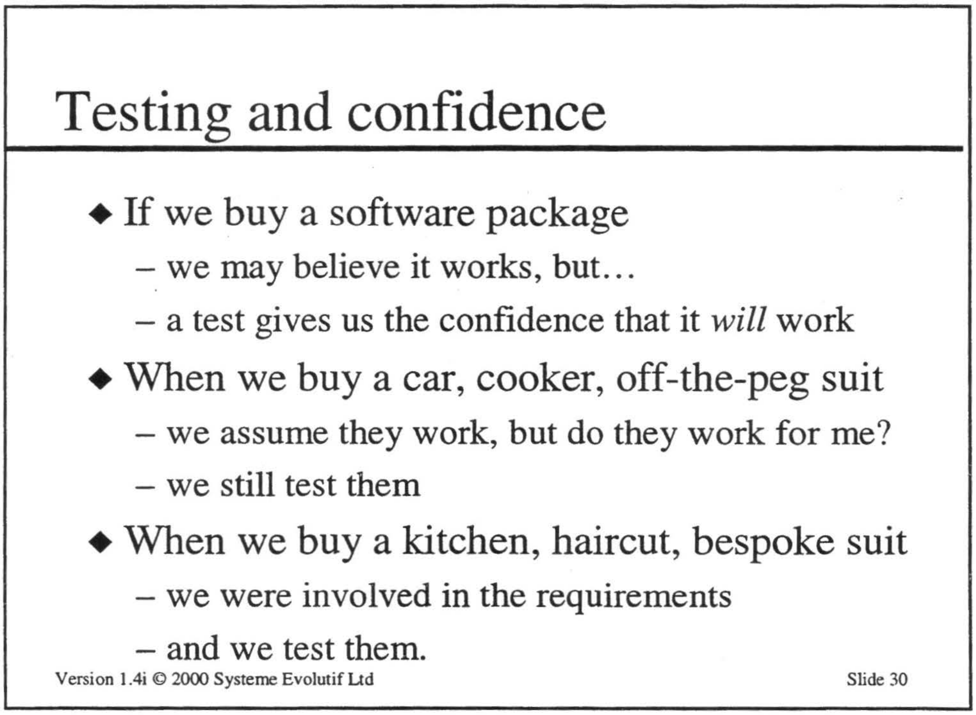
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 28



Testing and quality

- Testing measures quality
  - if we assess the rigour and number of tests and if we count the number of faults found...
  - we can make an objective assessment of the quality of the system under test
- Testing improves quality
  - tests aim to identify faults
  - if faults are found, we can improve the quality of the software.

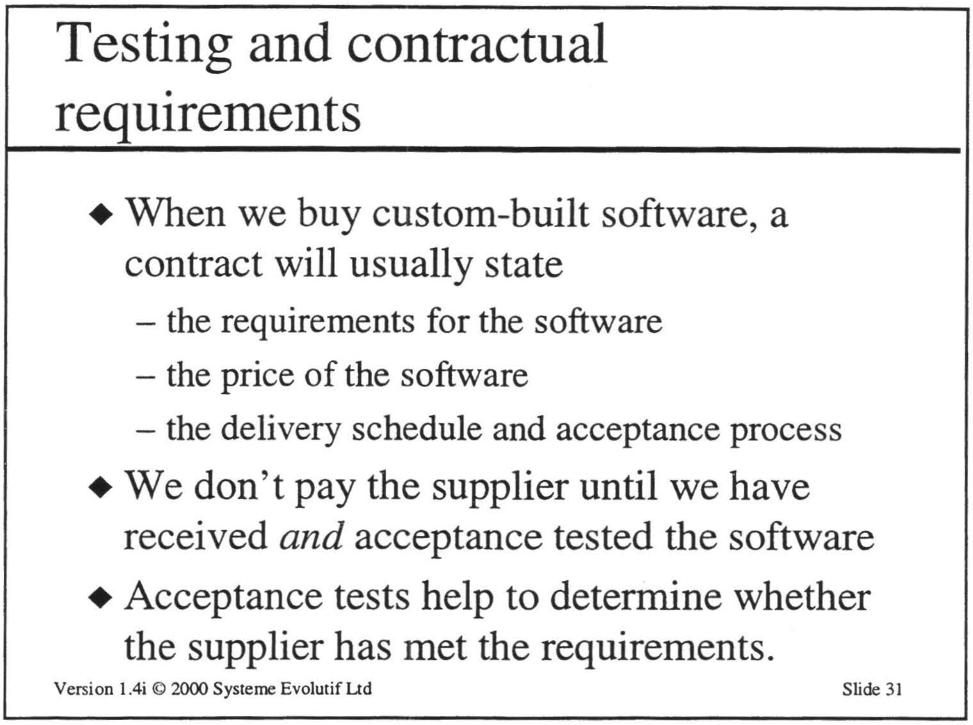
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 29



Testing and confidence

- If we buy a software package
  - we may believe it works, but…
  - a test gives us the confidence that it *will* work
- When we buy a car, cooker, off-the-peg suit
  - we assume they work, but do they work for me?
  - we still test them
- When we buy a kitchen, haircut, bespoke suit
  - we were involved in the requirements
  - and we test them.

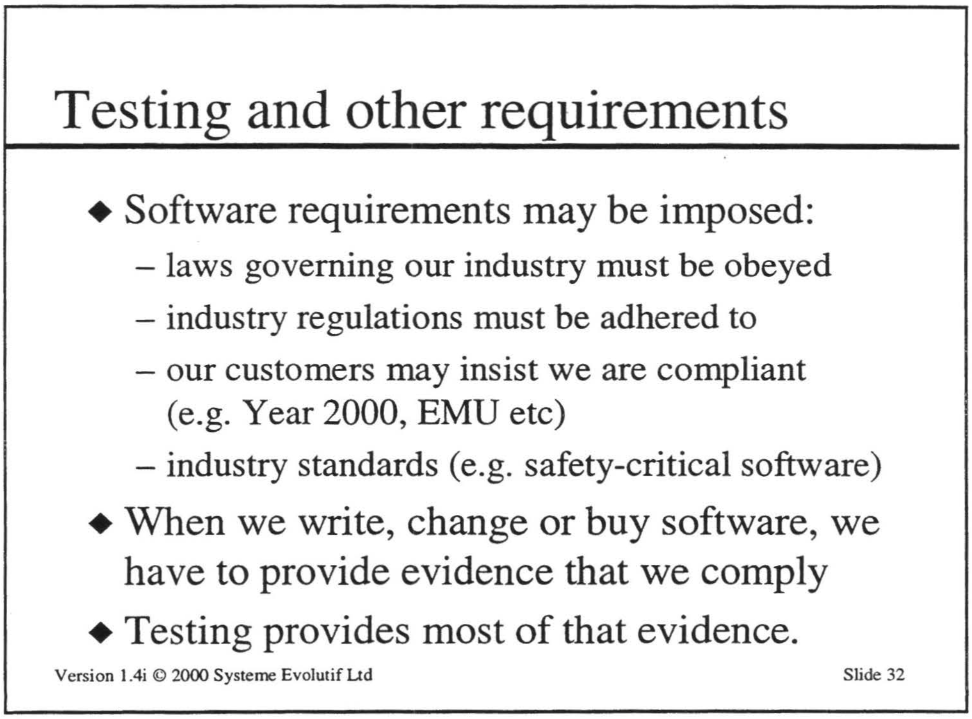
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 30



Testing and contractual requirements

- When we buy custom-built software, a contract will usually state
  - the requirements for the software
  - the price of the software
  - the delivery schedule and acceptance process
- We don't pay the supplier until we have received *and* acceptance tested the software
- Acceptance tests help to determine whether the supplier has met the requirements.

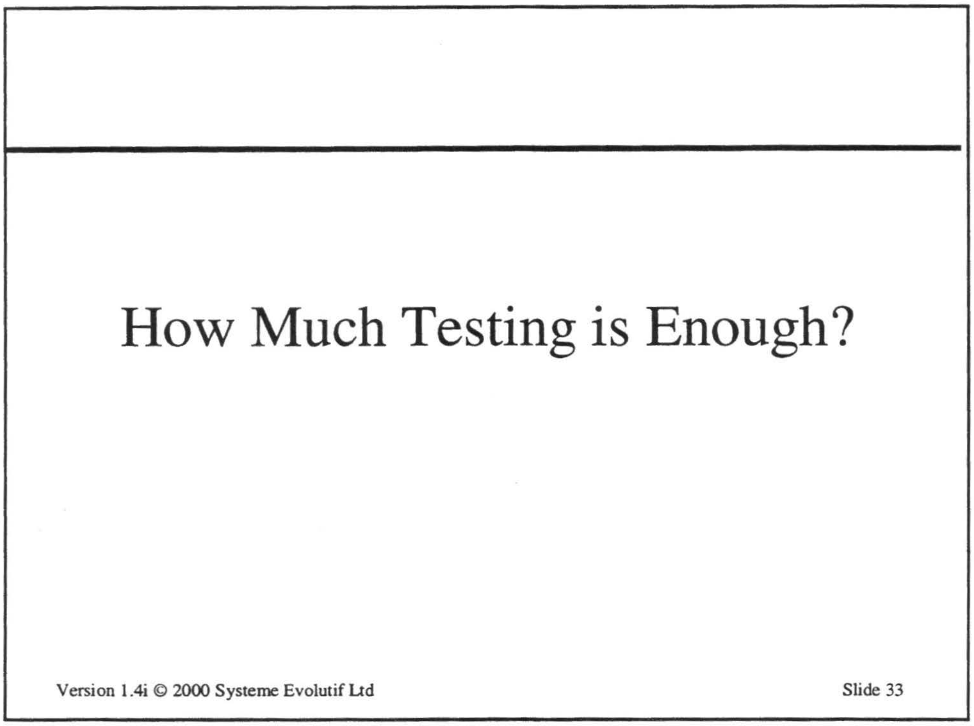
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 31



Testing and other requirements

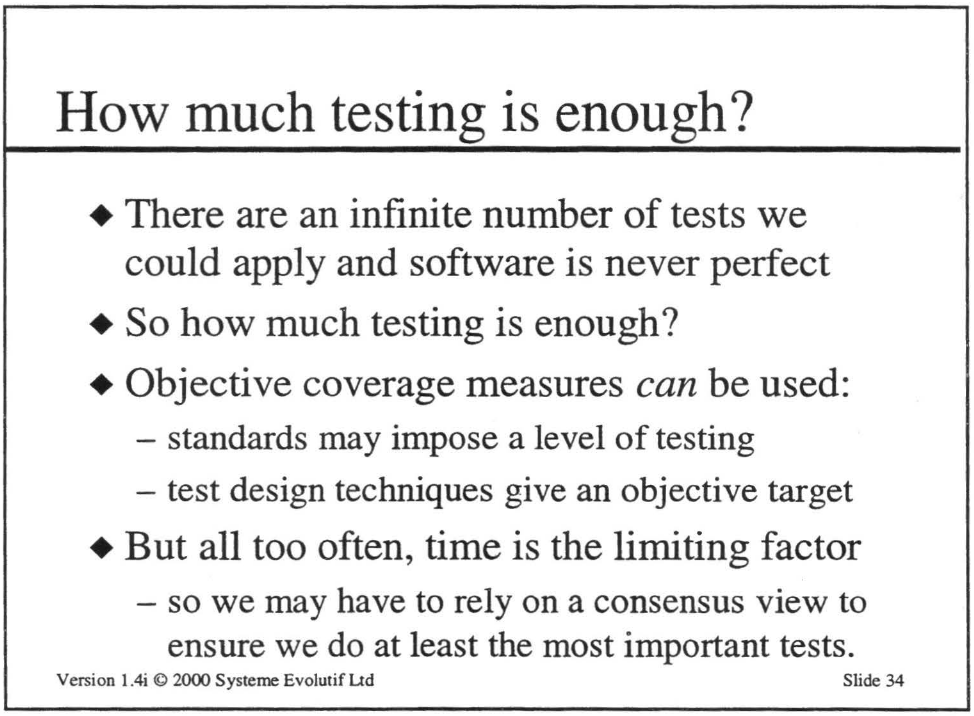
- Software requirements may be imposed:
  - laws governing our industry must be obeyed
  - industry regulations must be adhered to
  - our customers may insist we are compliant (e.g. Year 2000, EMU etc)
  - industry standards (e.g. safety-critical software)
- When we write, change or buy software, we have to provide evidence that we comply
- Testing provides most of that evidence.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 32



How Much Testing is Enough?

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 33

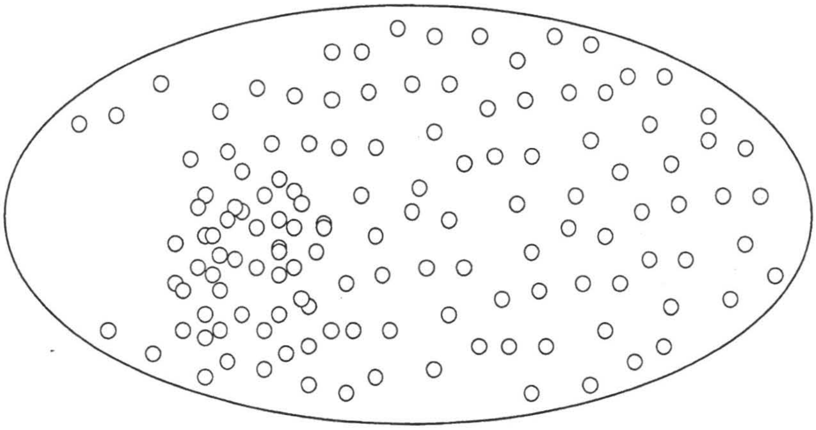


How much testing is enough?

- There are an infinite number of tests we could apply and software is never perfect
- So how much testing is enough?
- Objective coverage measures *can* be used:
  - standards may impose a level of testing
  - test design techniques give an objective target
- But all too often, time is the limiting factor
  - so we may have to rely on a consensus view to ensure we do at least the most important tests.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 34

The bugs that lurk in our systems



Where are the bugs?

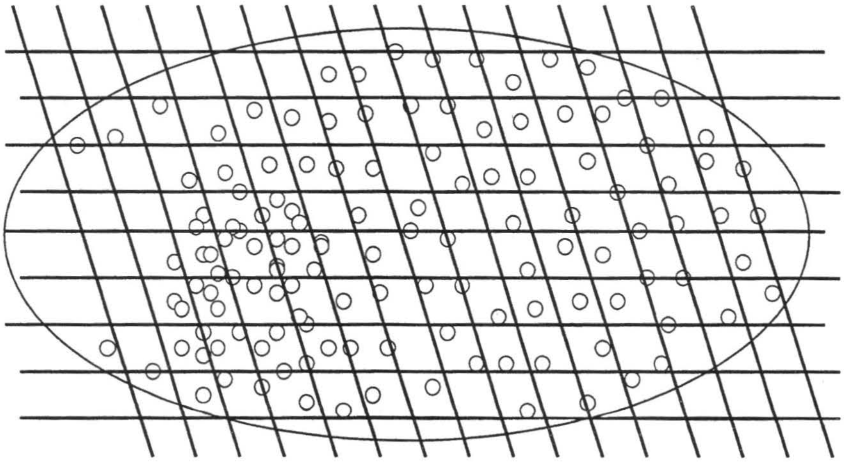
- ◆ Of course, if we knew that, we could fix them and go home!
- ◆ Experience tells us
  - – bugs are sociable! - they tend to cluster
  - – some parts of the system will be relatively bug-free (bought-in or unchanged code)
  - – bug fixing and maintenance are error-prone - 50% of changes cause other faults.

Testing as a fishing net

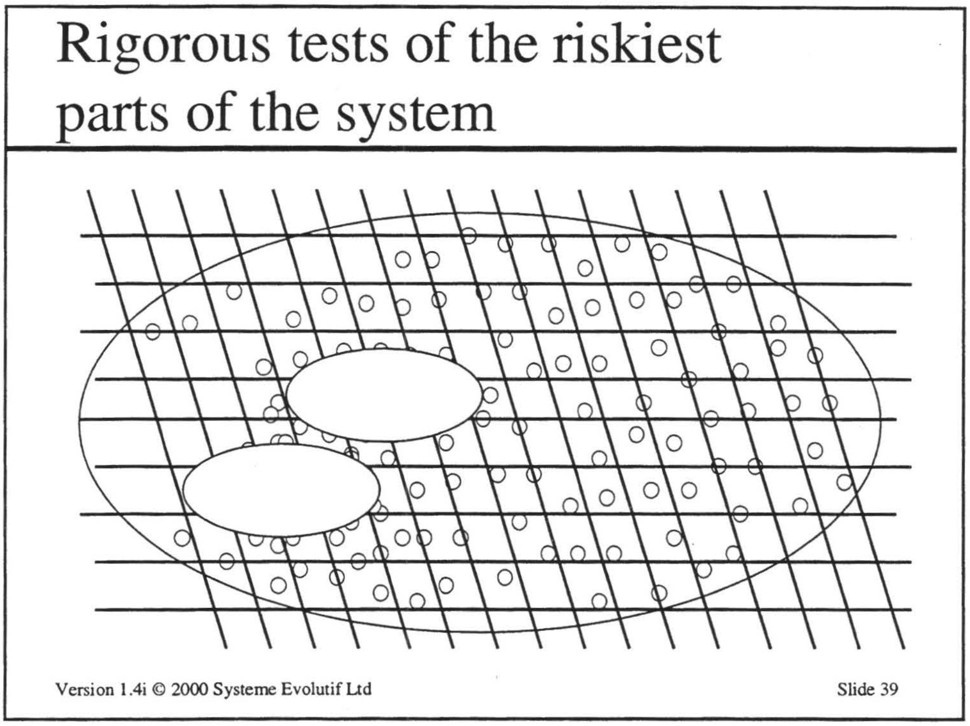
- ◆ Size of the net reflects size of the test
- ◆ Density of the mesh reflects test depth or thoroughness
- ◆ Large nets, large mesh to give overall confidence
- ◆ Small nets, fine mesh to find bugs
  - – use in areas that are critical to make sure the bugs aren’t there
  - – use in areas where we expect lots of bugs.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 37

Coverage of business-oriented tests



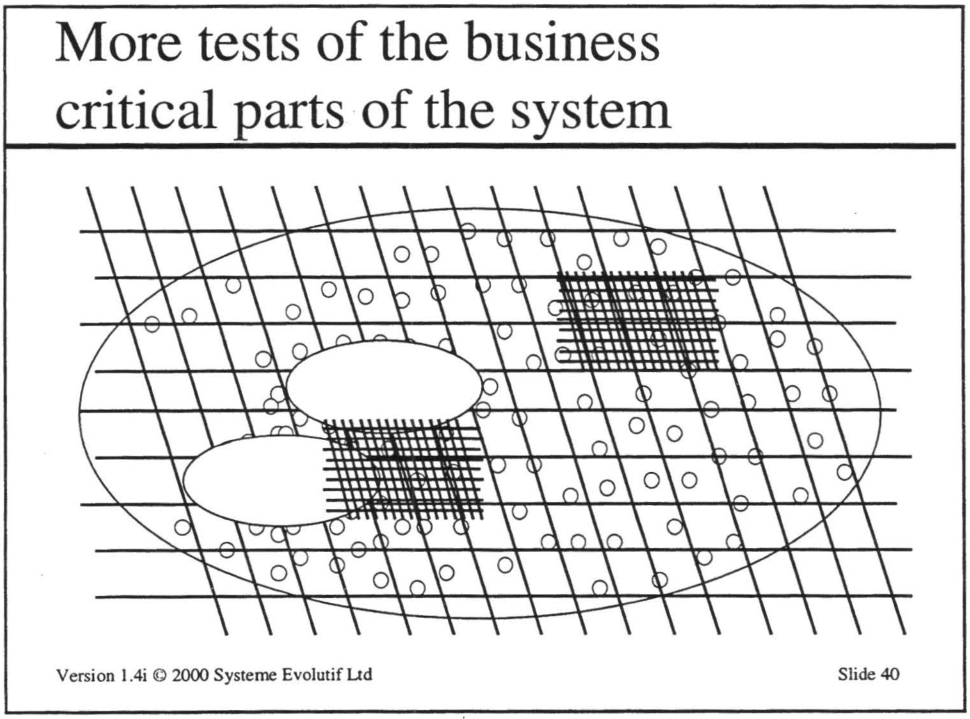
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 38



Rigorous tests of the riskiest parts of the system

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 39

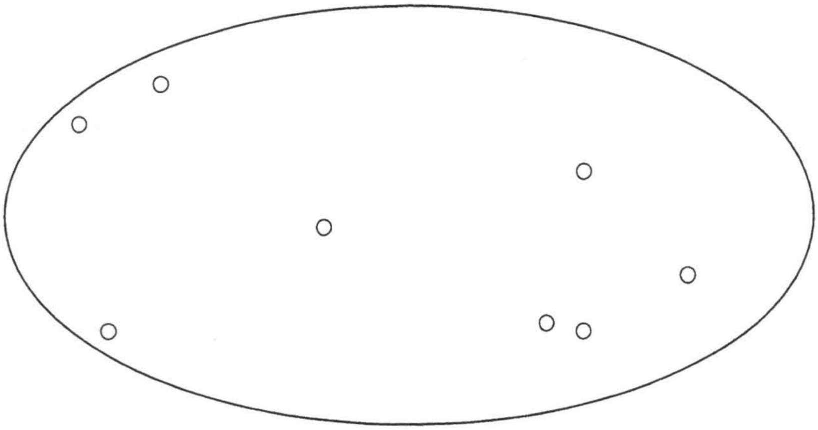


More tests of the business critical parts of the system

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 40

What about the bugs we don’t find?

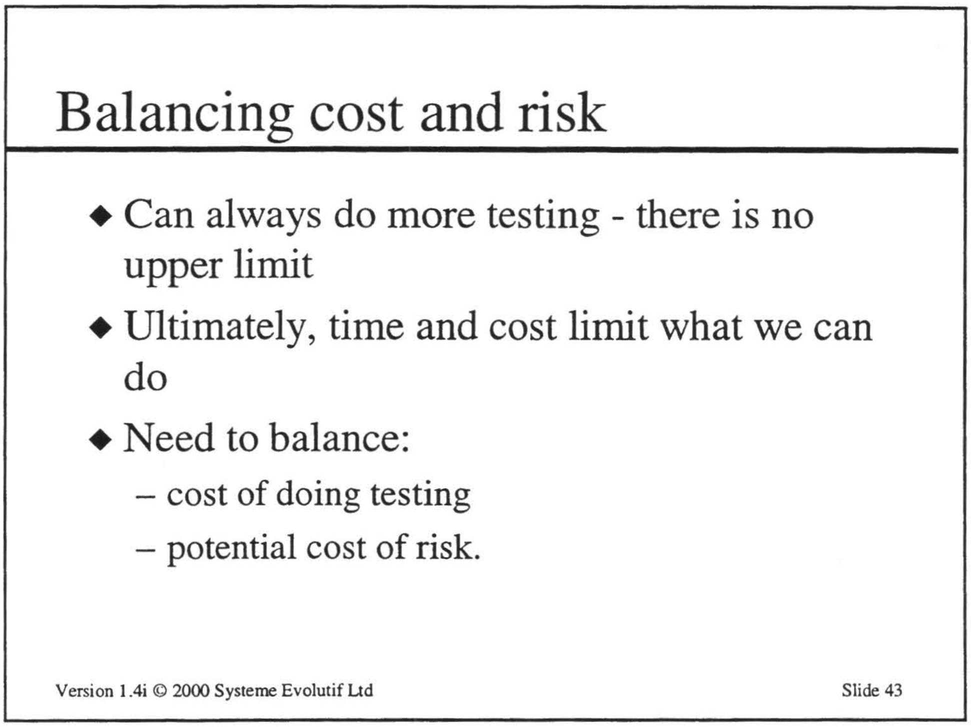


Version 1.4i © 2000 Systeme Evolutif Ltd Slide 41

What about the bugs we don’t find?

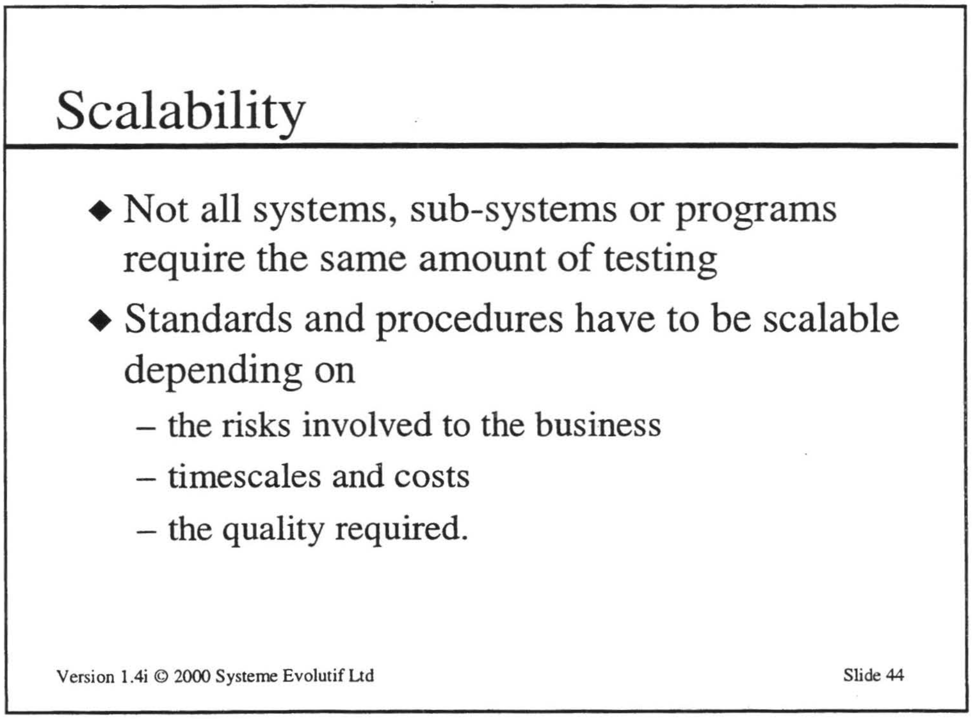
- If not in the business critical parts of the system - would the users care?
- If not in the system critical parts of the system - should be low impact
- If they are in the critical parts of the system
  - should be few and far between
  - should be very obscure.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 42



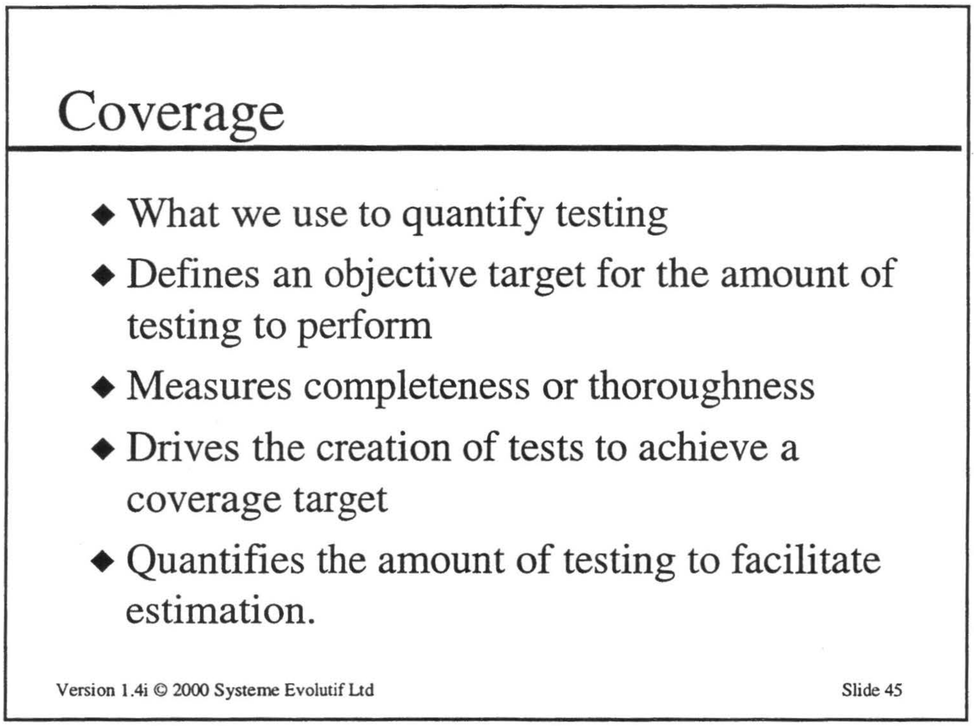
Balancing cost and risk

- Can always do more testing - there is no upper limit
- Ultimately, time and cost limit what we can do
- Need to balance:
  - cost of doing testing
  - potential cost of risk.



Scalability

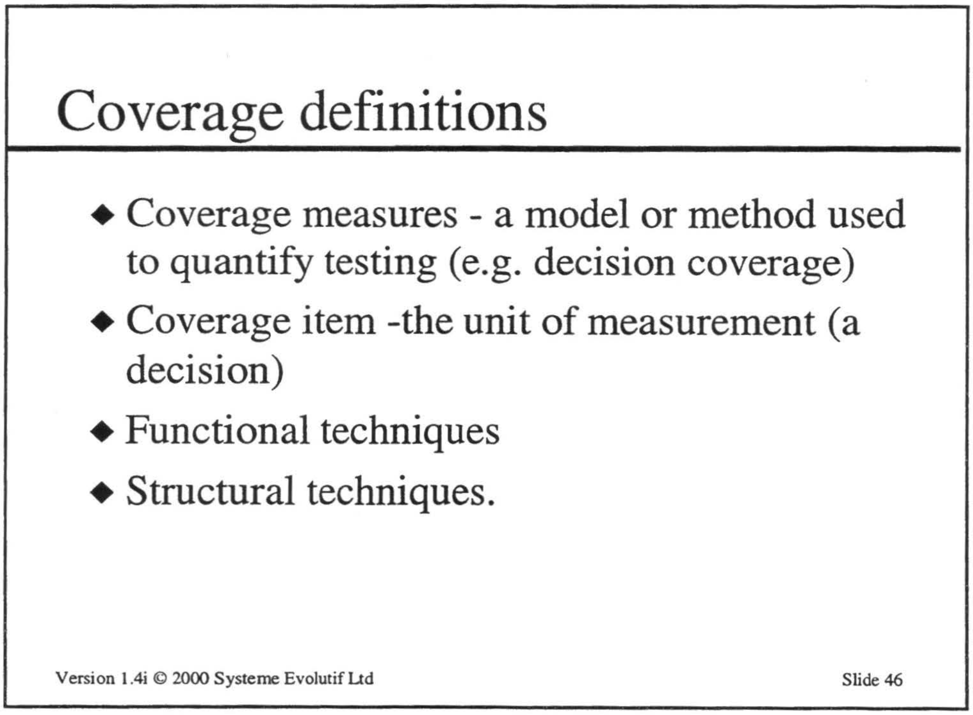
- Not all systems, sub-systems or programs require the same amount of testing
- Standards and procedures have to be scalable depending on
  - the risks involved to the business
  - timescales and costs
  - the quality required.



Coverage

- What we use to quantify testing
- Defines an objective target for the amount of testing to perform
- Measures completeness or thoroughness
- Drives the creation of tests to achieve a coverage target
- Quantifies the amount of testing to facilitate estimation.

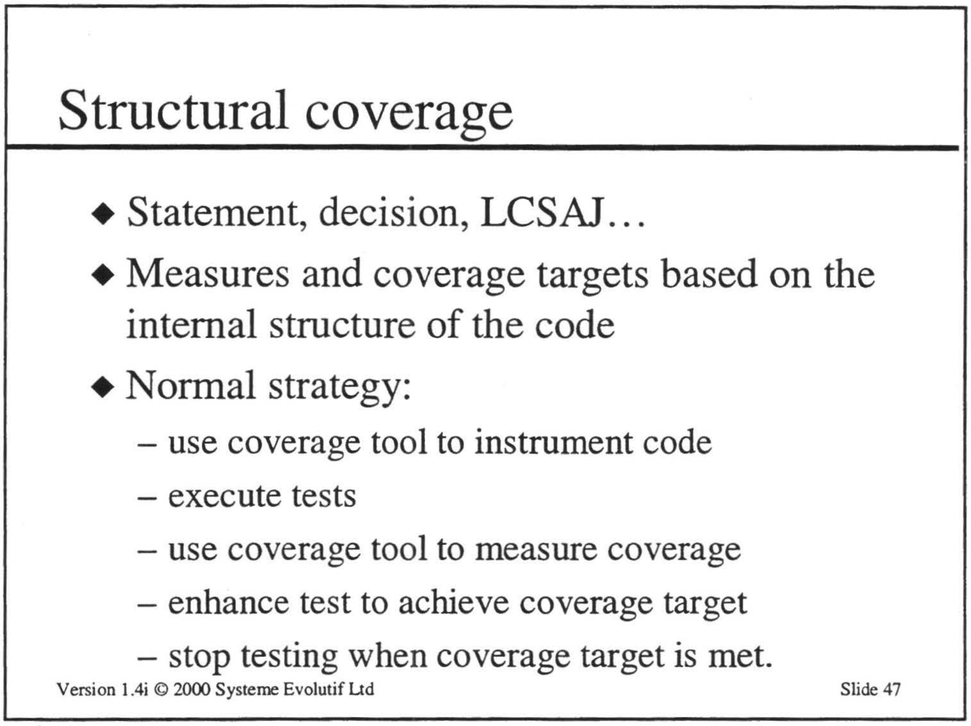
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 45



Coverage definitions

- Coverage measures - a model or method used to quantify testing (e.g. decision coverage)
- Coverage item -the unit of measurement (a decision)
- Functional techniques
- Structural techniques.

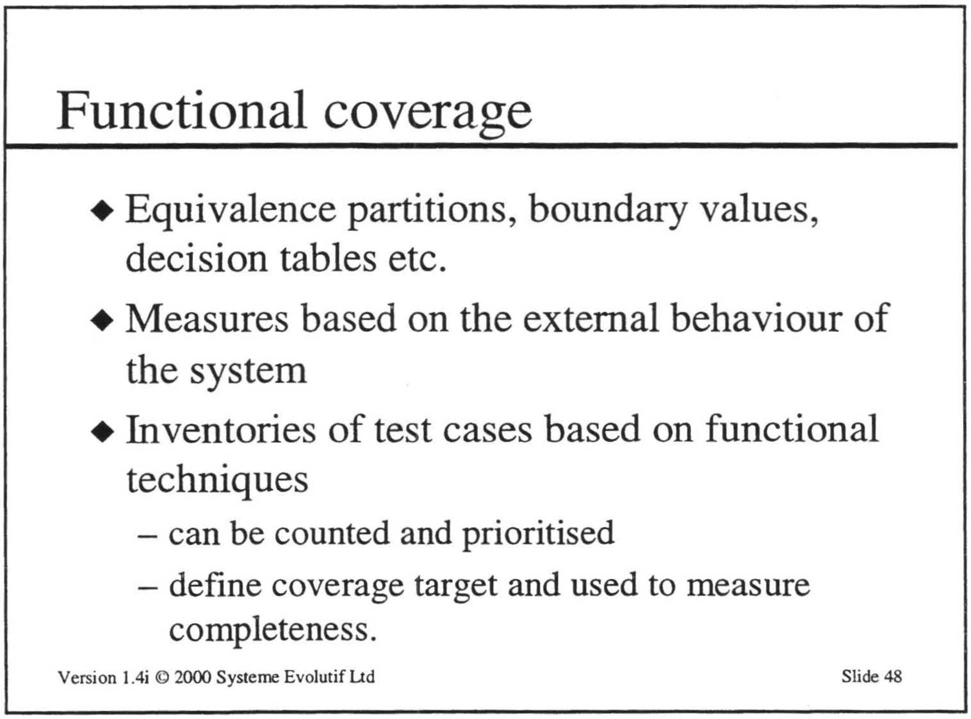
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 46



Structural coverage

- Statement, decision, LCSAJ…
- Measures and coverage targets based on the internal structure of the code
- Normal strategy:
  - use coverage tool to instrument code
  - execute tests
  - use coverage tool to measure coverage
  - enhance test to achieve coverage target
  - stop testing when coverage target is met.

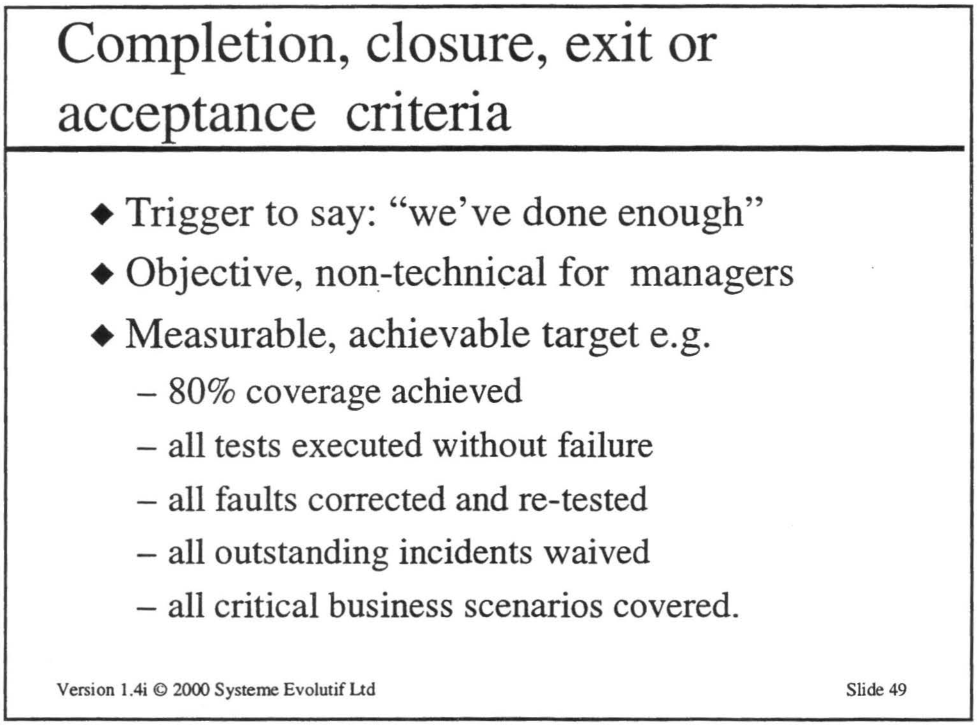
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 47



Functional coverage

- Equivalence partitions, boundary values, decision tables etc.
- Measures based on the external behaviour of the system
- Inventories of test cases based on functional techniques
  - can be counted and prioritised
  - define coverage target and used to measure completeness.

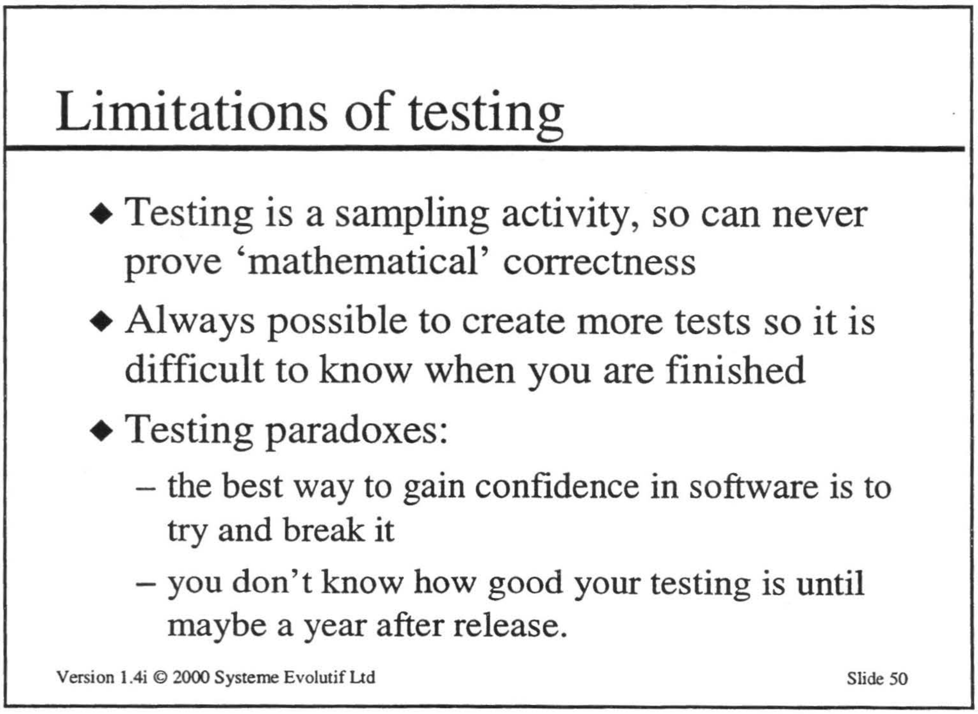
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 48



Completion, closure, exit or acceptance criteria

- ◆ Trigger to say: "we've done enough"
- ◆ Objective, non-technical for managers
- ◆ Measurable, achievable target e.g.
  - – 80% coverage achieved
  - – all tests executed without failure
  - – all faults corrected and re-tested
  - – all outstanding incidents waived
  - – all critical business scenarios covered.

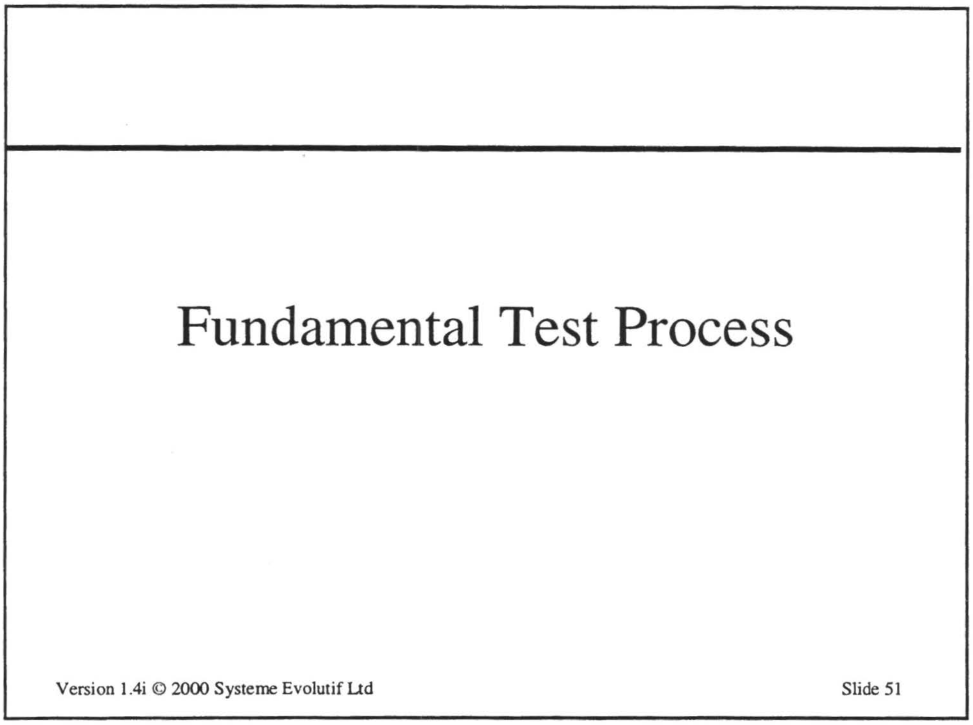
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 49



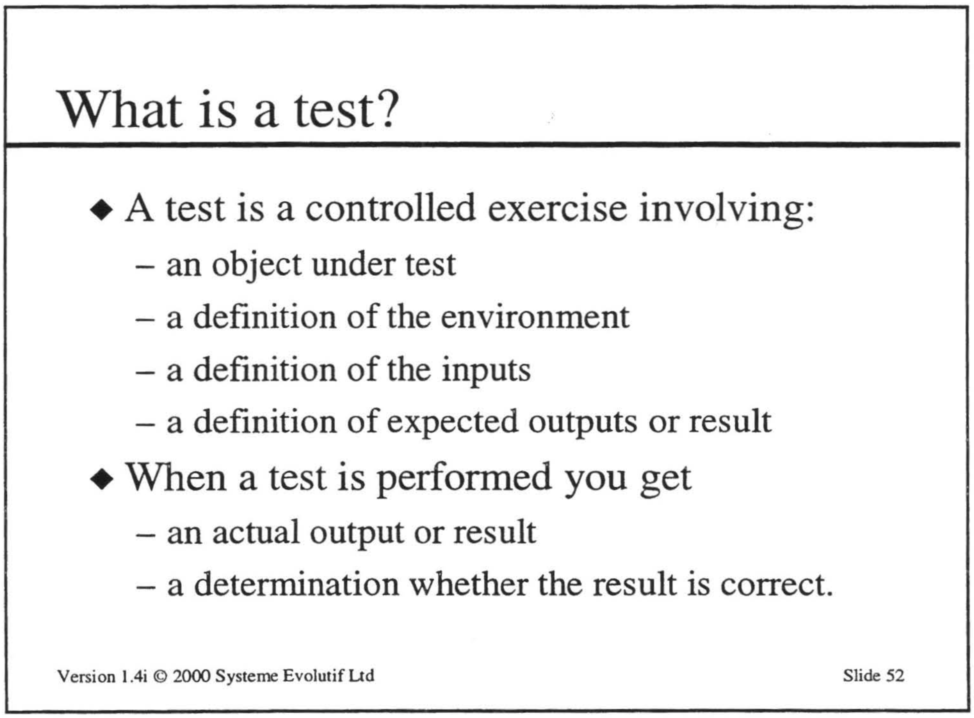
Limitations of testing

- ◆ Testing is a sampling activity, so can never prove 'mathematical' correctness
- ◆ Always possible to create more tests so it is difficult to know when you are finished
- ◆ Testing paradoxes:
  - – the best way to gain confidence in software is to try and break it
  - – you don't know how good your testing is until maybe a year after release.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 50

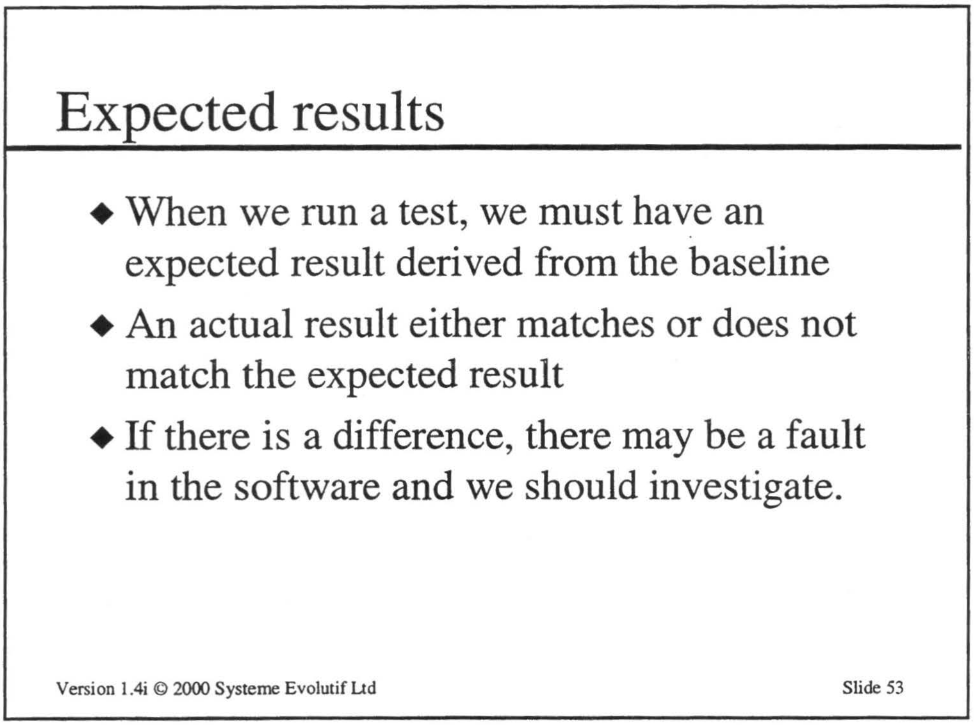


Fundamental Test Process



What is a test?

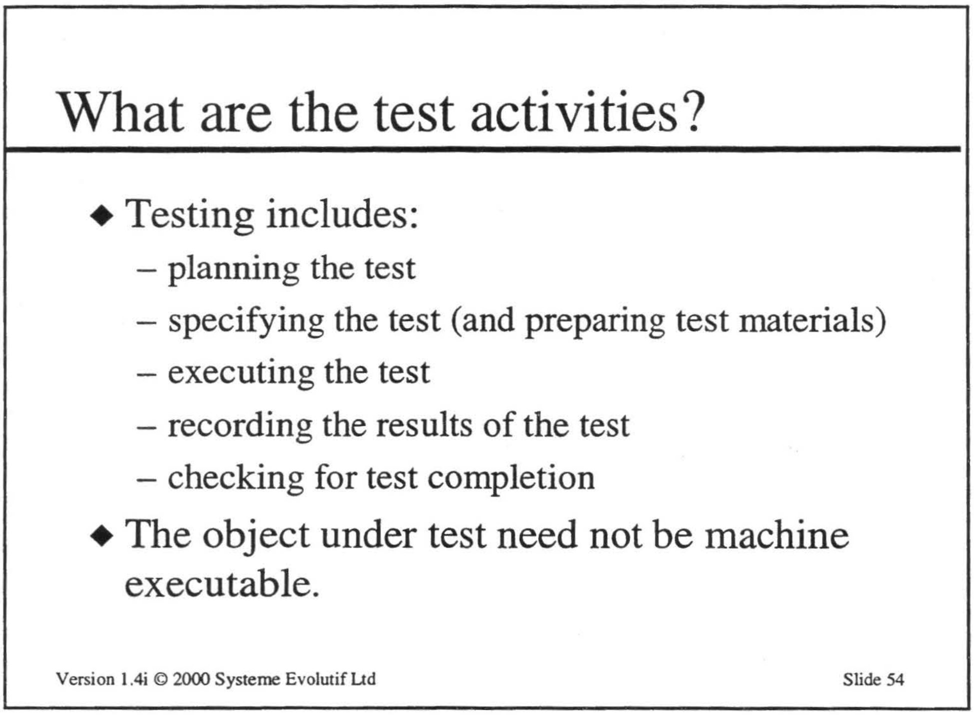
- A test is a controlled exercise involving:
  - – an object under test
  - – a definition of the environment
  - – a definition of the inputs
  - – a definition of expected outputs or result
- When a test is performed you get
  - – an actual output or result
  - – a determination whether the result is correct.



Expected results

- ◆ When we run a test, we must have an expected result derived from the baseline
- ◆ An actual result either matches or does not match the expected result
- ◆ If there is a difference, there may be a fault in the software and we should investigate.

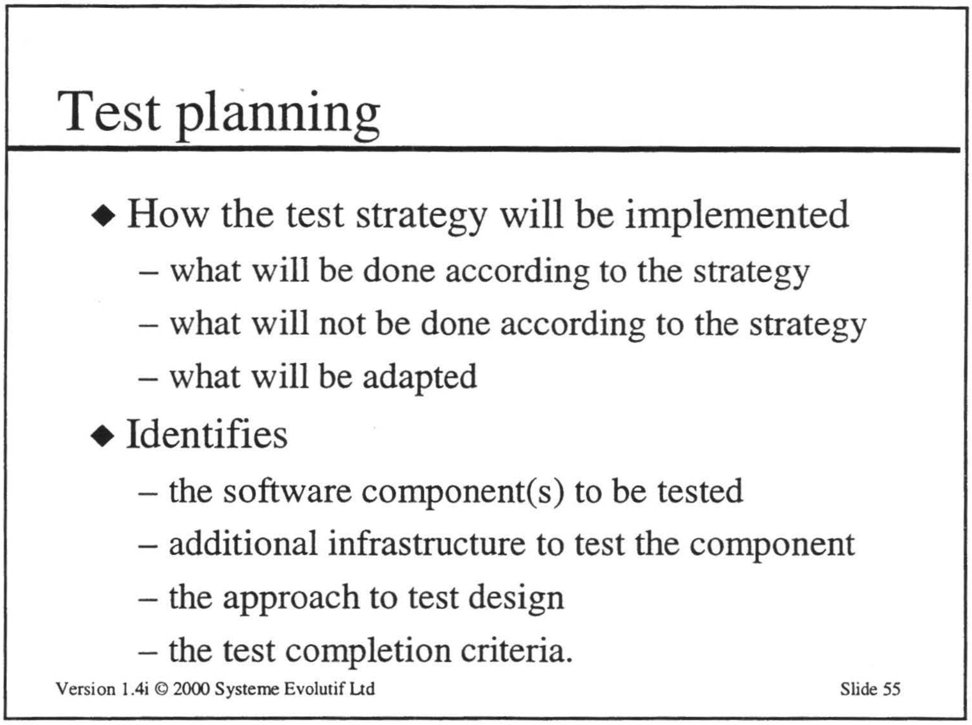
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 53



What are the test activities?

- ◆ Testing includes:
  - – planning the test
  - – specifying the test (and preparing test materials)
  - – executing the test
  - – recording the results of the test
  - – checking for test completion
- ◆ The object under test need not be machine executable.

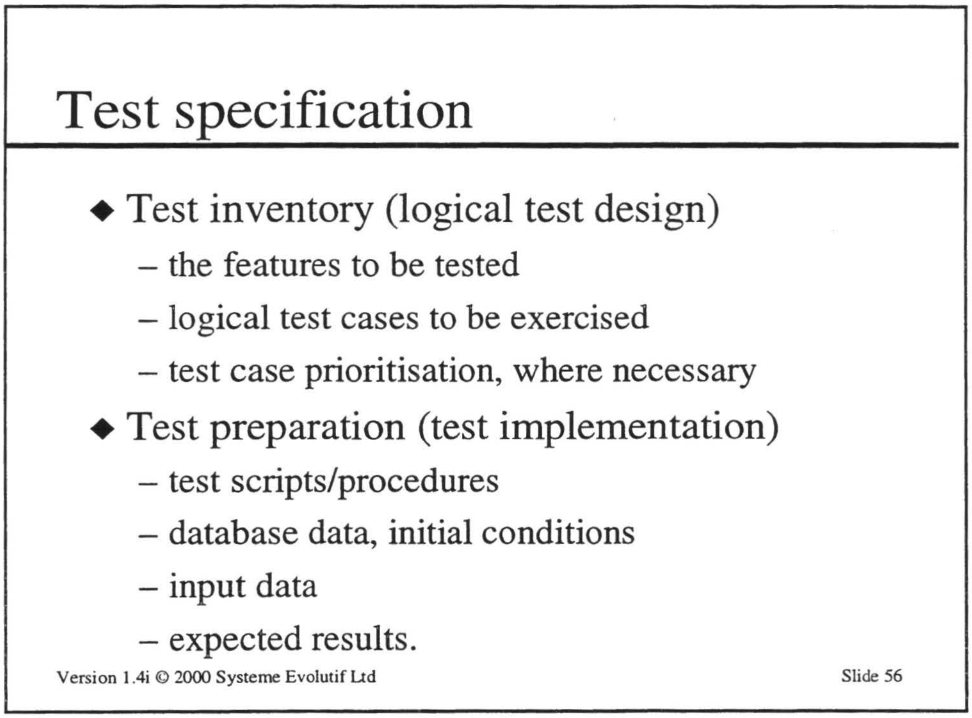
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 54



Test planning

- How the test strategy will be implemented
  - what will be done according to the strategy
  - what will not be done according to the strategy
  - what will be adapted
- Identifies
  - the software component(s) to be tested
  - additional infrastructure to test the component
  - the approach to test design
  - the test completion criteria.

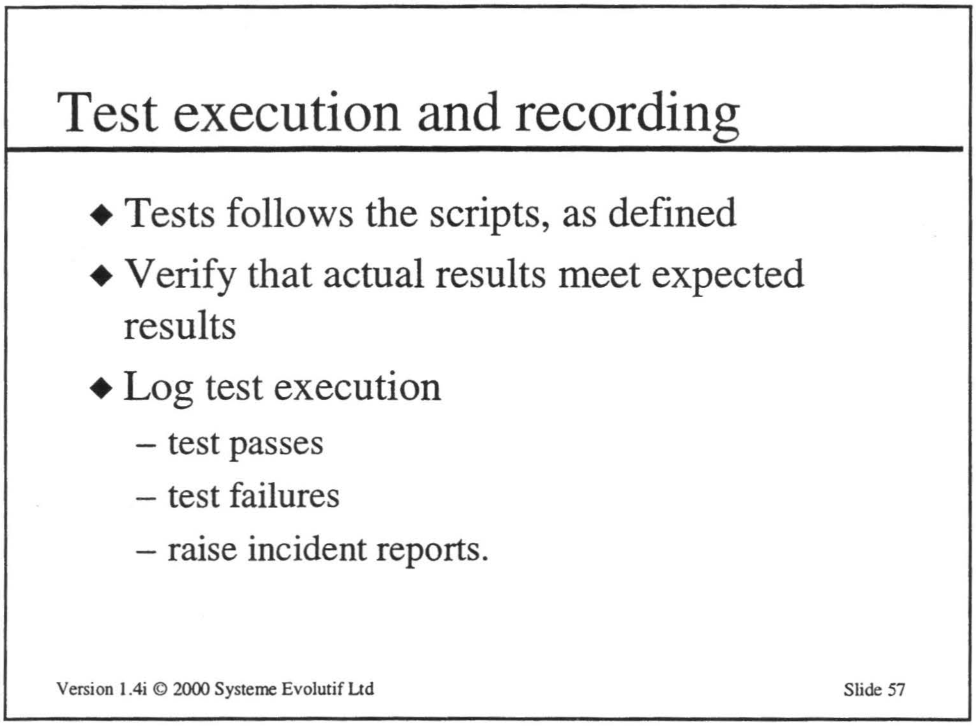
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 55



Test specification

- Test inventory (logical test design)
  - the features to be tested
  - logical test cases to be exercised
  - test case prioritisation, where necessary
- Test preparation (test implementation)
  - test scripts/procedures
  - database data, initial conditions
  - input data
  - expected results.

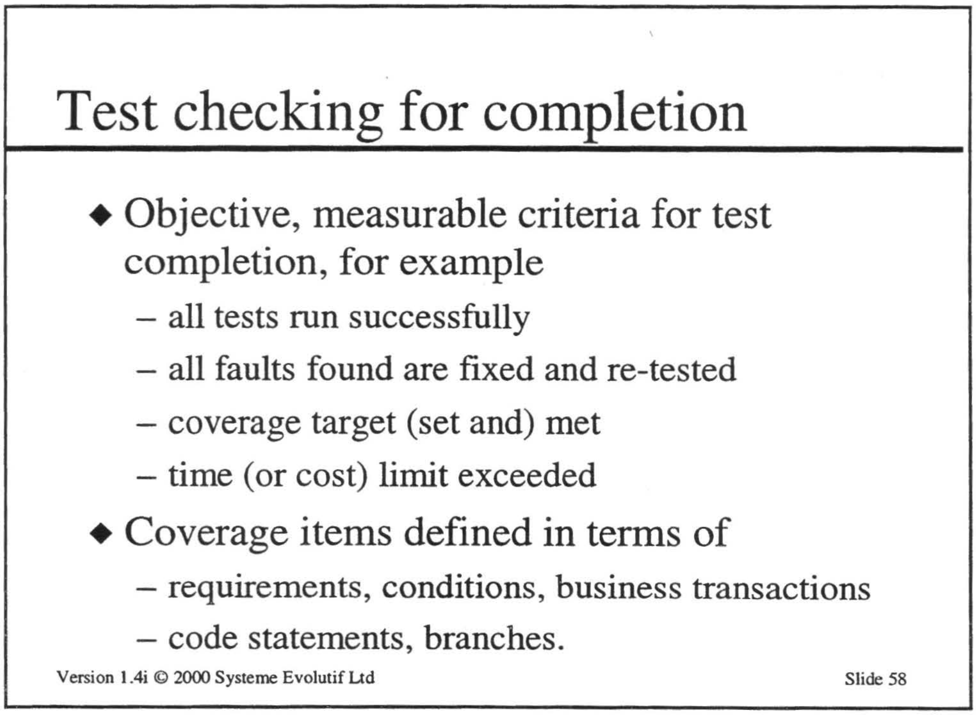
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 56



Test execution and recording

- Tests follows the scripts, as defined
- Verify that actual results meet expected results
- Log test execution
  - test passes
  - test failures
  - raise incident reports.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 57



Test checking for completion

- Objective, measurable criteria for test completion, for example
  - all tests run successfully
  - all faults found are fixed and re-tested
  - coverage target (set and) met
  - time (or cost) limit exceeded
- Coverage items defined in terms of
  - requirements, conditions, business transactions
  - code statements, branches.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 58

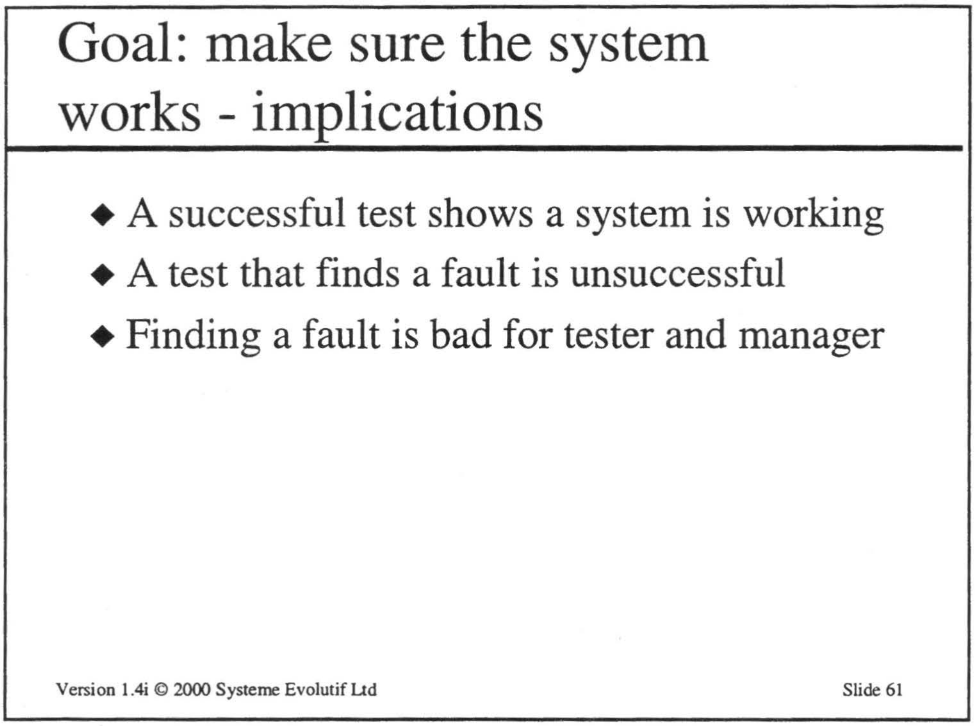
Test checking for completion (2)

- Under time pressure in low integrity systems
  - some faults may be acceptable (for this release)
  - some tests may not be run at all
- If there are no tests left, but there is still time
  - maybe some additional tests could be run
- You may decide to release the software now, but testing could continue.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 59

The Psychology of Testing

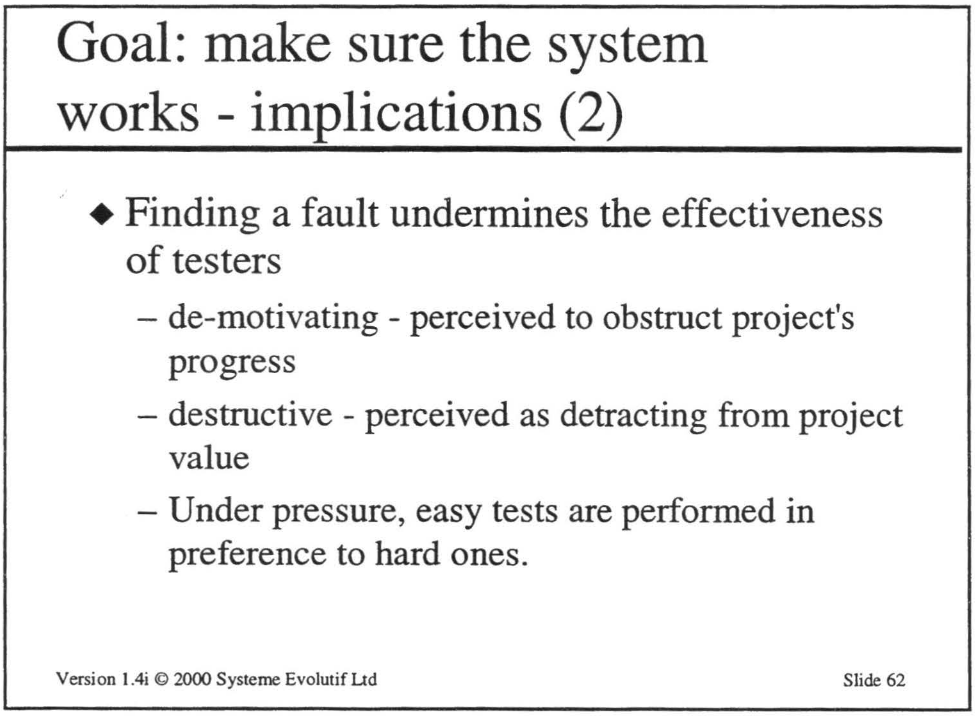
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 60



Goal: make sure the system works - implications

- A successful test shows a system is working
- A test that finds a fault is unsuccessful
- Finding a fault is bad for tester and manager

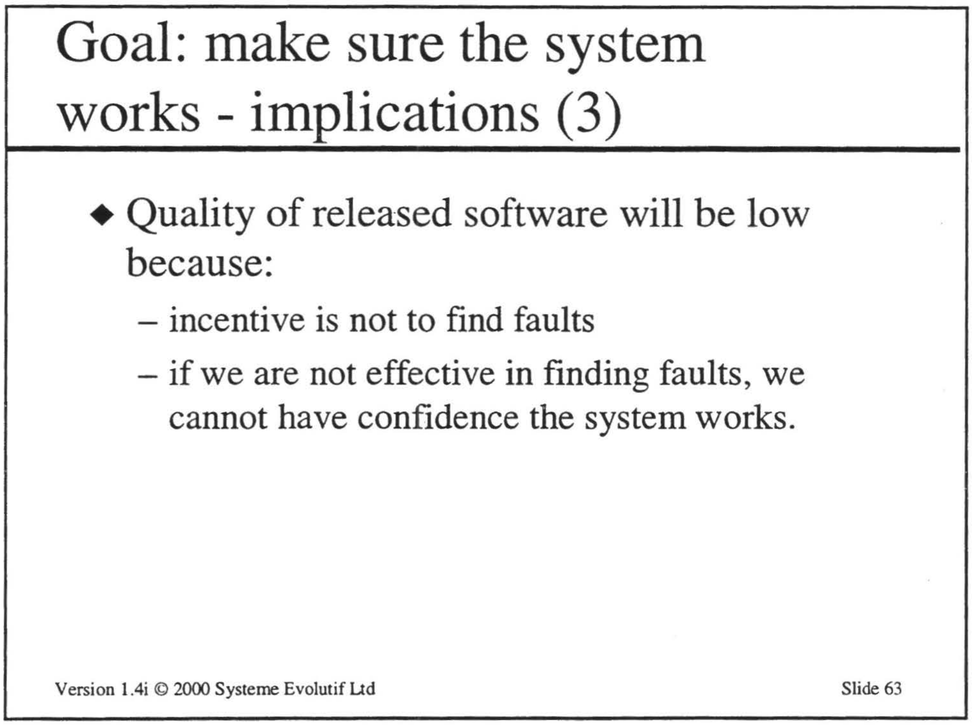
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 61



Goal: make sure the system works - implications (2)

- Finding a fault undermines the effectiveness of testers
  - de-motivating - perceived to obstruct project's progress
  - destructive - perceived as detracting from project value
  - Under pressure, easy tests are performed in preference to hard ones.

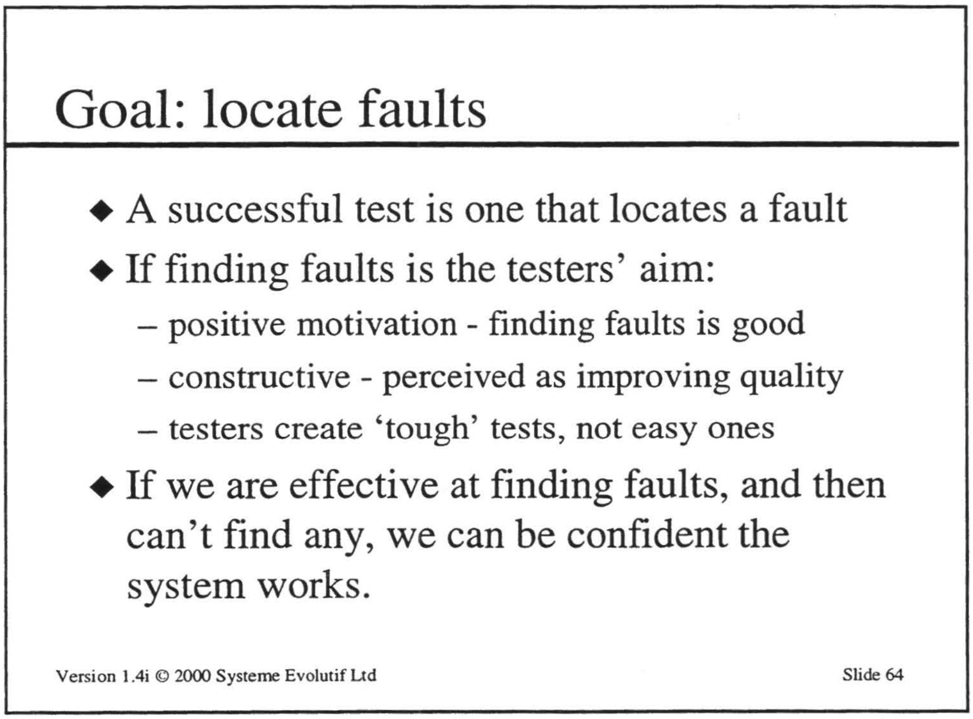
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 62



Goal: make sure the system works - implications (3)

- Quality of released software will be low because:
  - incentive is not to find faults
  - if we are not effective in finding faults, we cannot have confidence the system works.

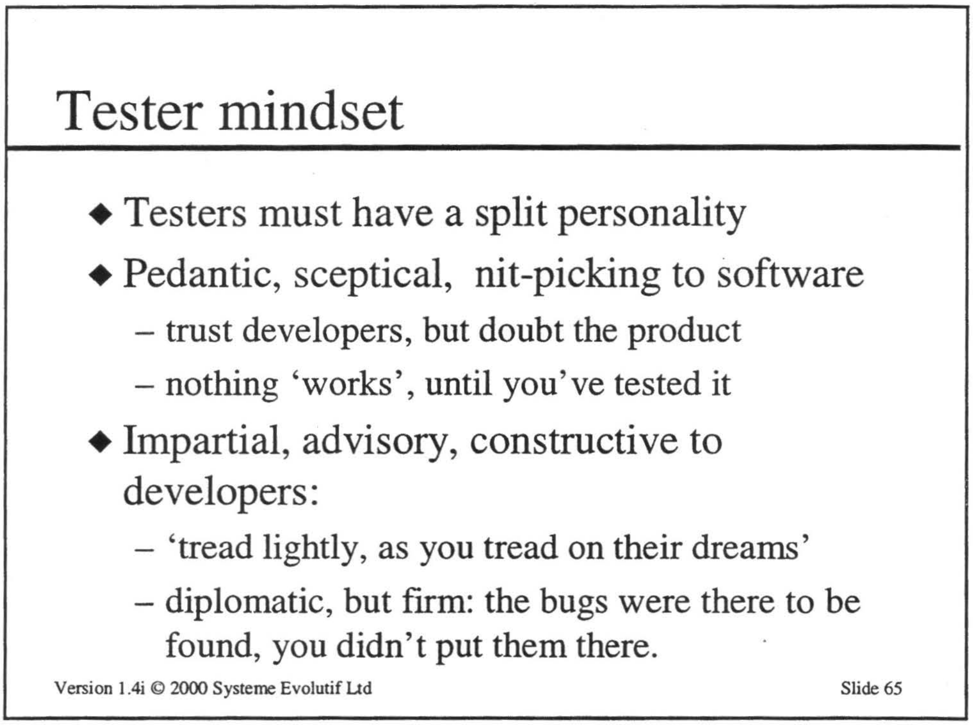
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 63



Goal: locate faults

- A successful test is one that locates a fault
- If finding faults is the testers' aim:
  - positive motivation - finding faults is good
  - constructive - perceived as improving quality
  - testers create 'tough' tests, not easy ones
- If we are effective at finding faults, and then can't find any, we can be confident the system works.

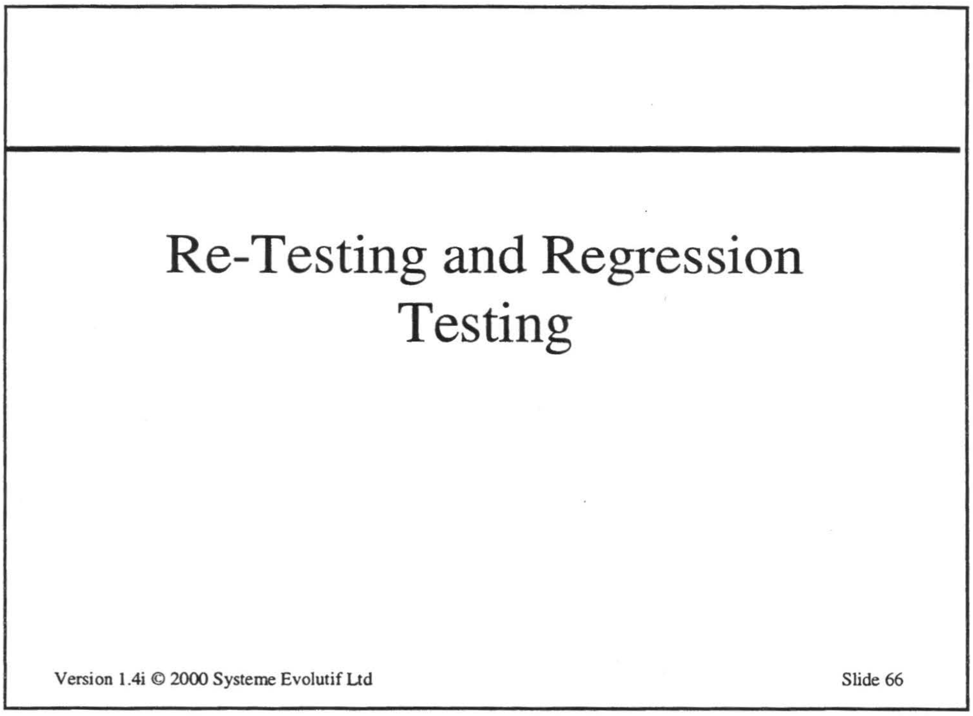
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 64



Tester mindset

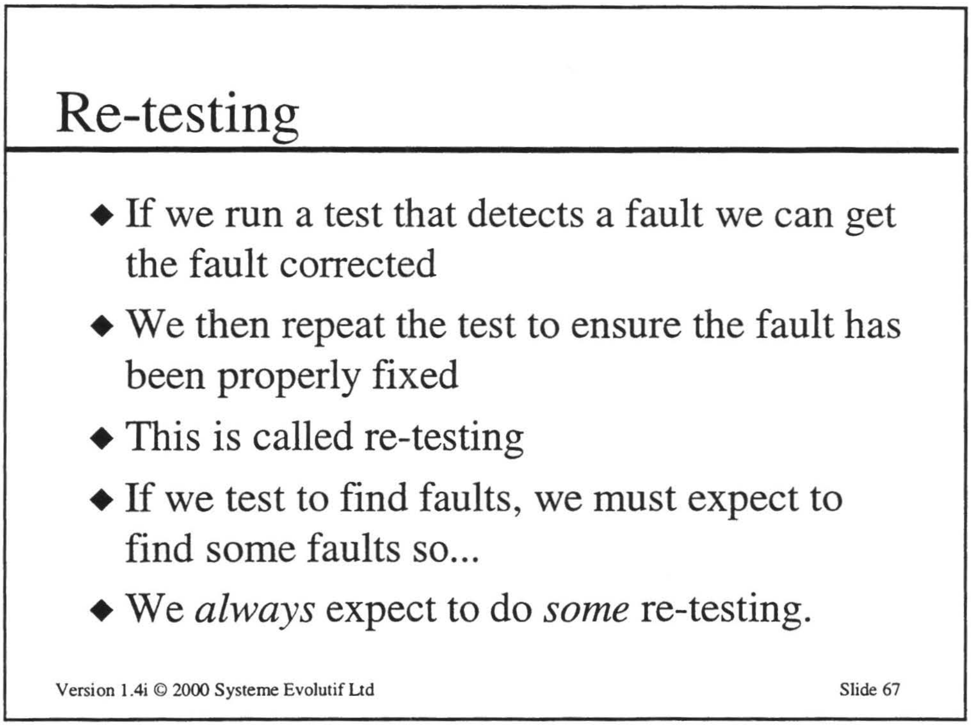
- Testers must have a split personality
- Pedantic, sceptical, nit-picking to software
  - trust developers, but doubt the product
  - nothing ‘works’, until you’ve tested it
- Impartial, advisory, constructive to developers:
  - ‘tread lightly, as you tread on their dreams’
  - diplomatic, but firm: the bugs were there to be found, you didn’t put them there.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 65



Re-Testing and Regression Testing

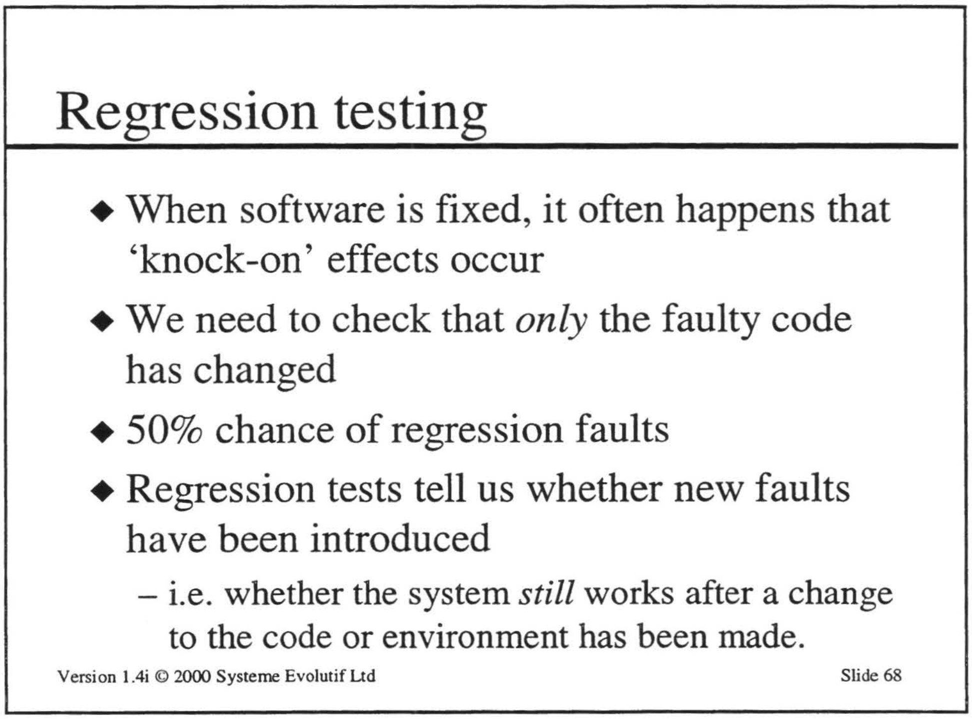
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 66



Re-testing

- ◆ If we run a test that detects a fault we can get the fault corrected
- ◆ We then repeat the test to ensure the fault has been properly fixed
- ◆ This is called re-testing
- ◆ If we test to find faults, we must expect to find some faults so...
- ◆ We *always* expect to do *some* re-testing.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 67



Regression testing

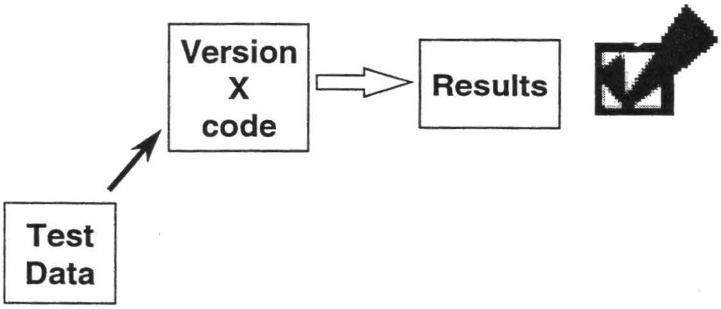
- ◆ When software is fixed, it often happens that 'knock-on' effects occur
- ◆ We need to check that *only* the faulty code has changed
- ◆ 50% chance of regression faults
- ◆ Regression tests tell us whether new faults have been introduced
  - – i.e. whether the system *still* works after a change to the code or environment has been made.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 68

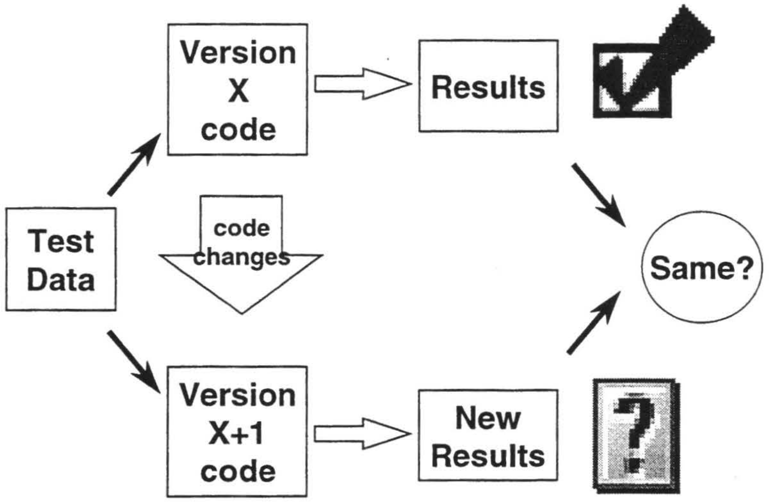
Regression testing (2)

- “Testing to ensure a change has not caused faults in unchanged parts of the system”
- Not necessarily a separate stage
- Regression testing most important during maintenance activities
- Effective regression testing is almost always automated.

Initial test failed to find a fault

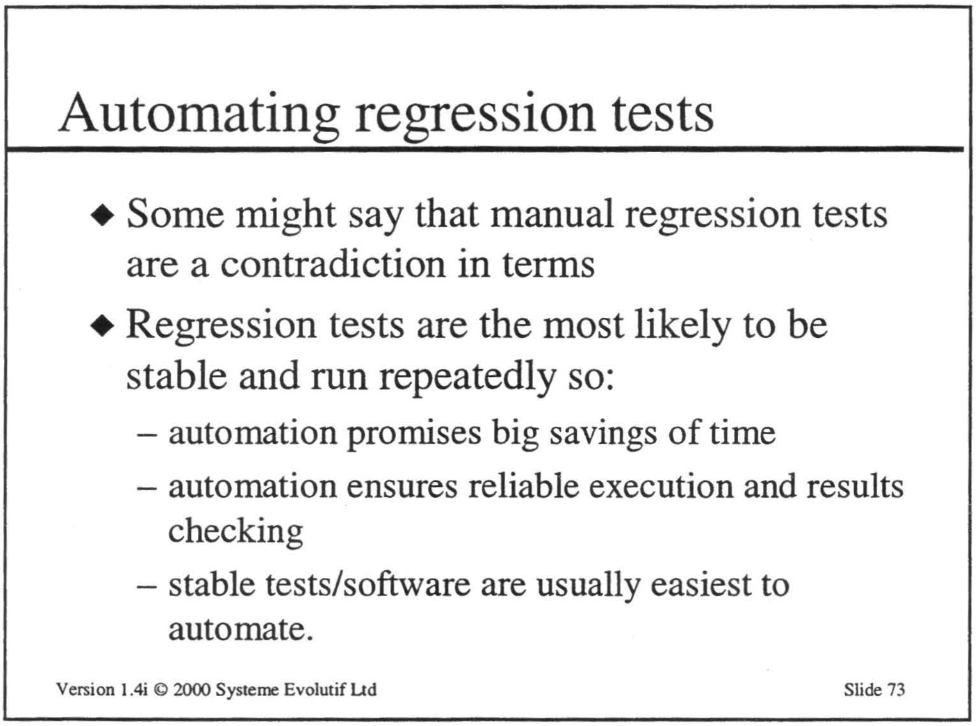


Regression test checks that the software still works



Selective regression tests

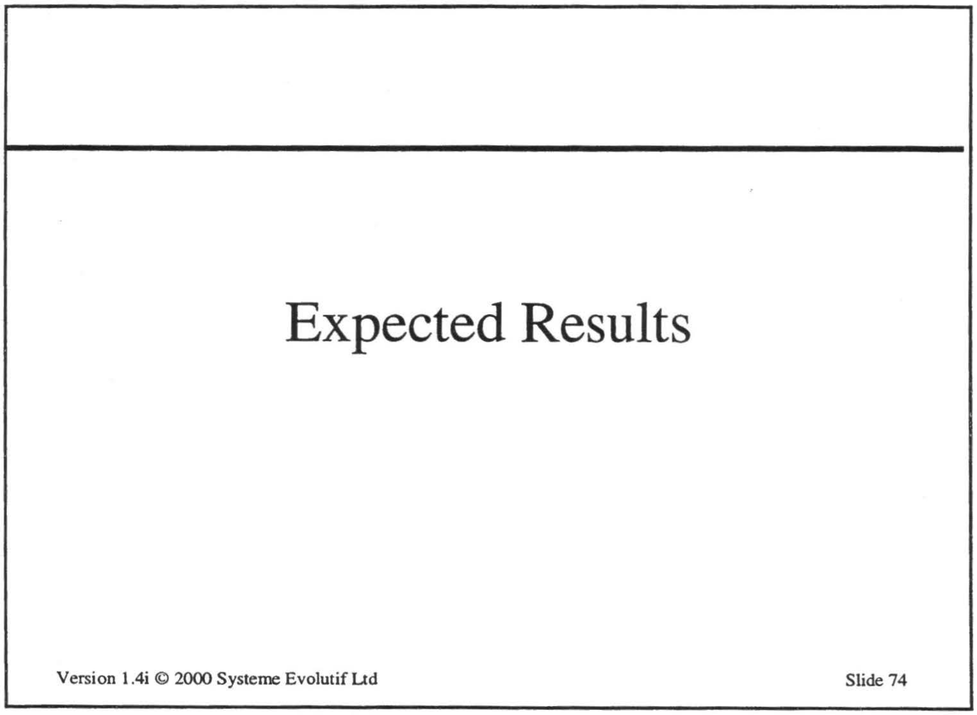
- An entire test may be retained for subsequent use as a regression test pack
- This may be uneconomic or impractical
  - complicated, expensive tests may drop out
  - a sub-set (say 20%) for emergency fixes
- Regression tests should be selected to:
  - exercise key functionality
  - cover the most error-prone areas of the software
  - be most easily automated.



Automating regression tests

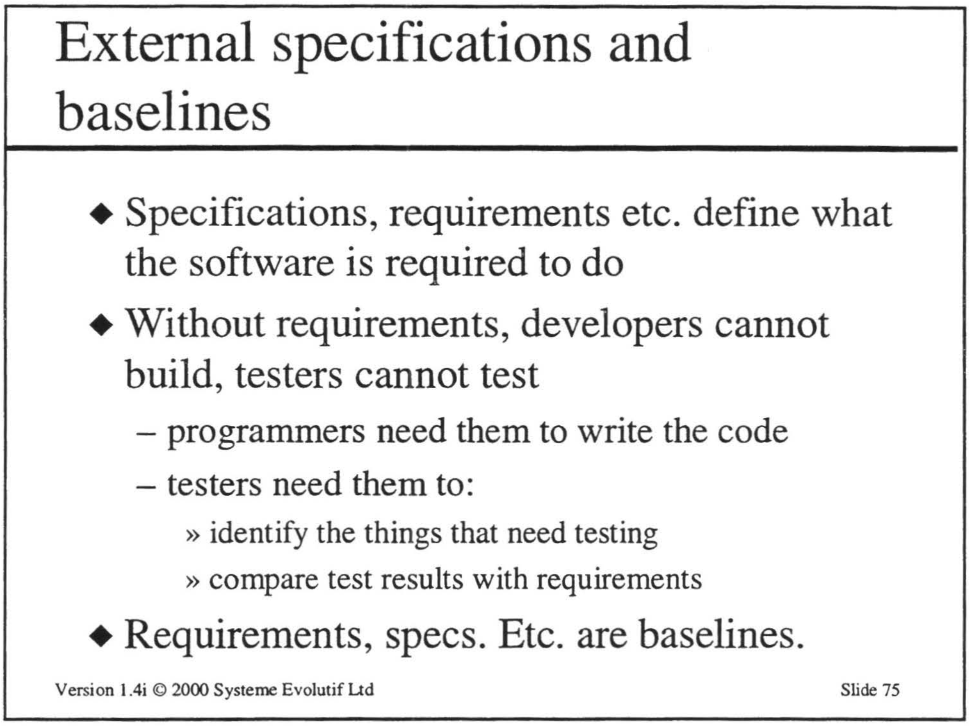
- Some might say that manual regression tests are a contradiction in terms
- Regression tests are the most likely to be stable and run repeatedly so:
  - automation promises big savings of time
  - automation ensures reliable execution and results checking
  - stable tests/software are usually easiest to automate.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 73



Expected Results

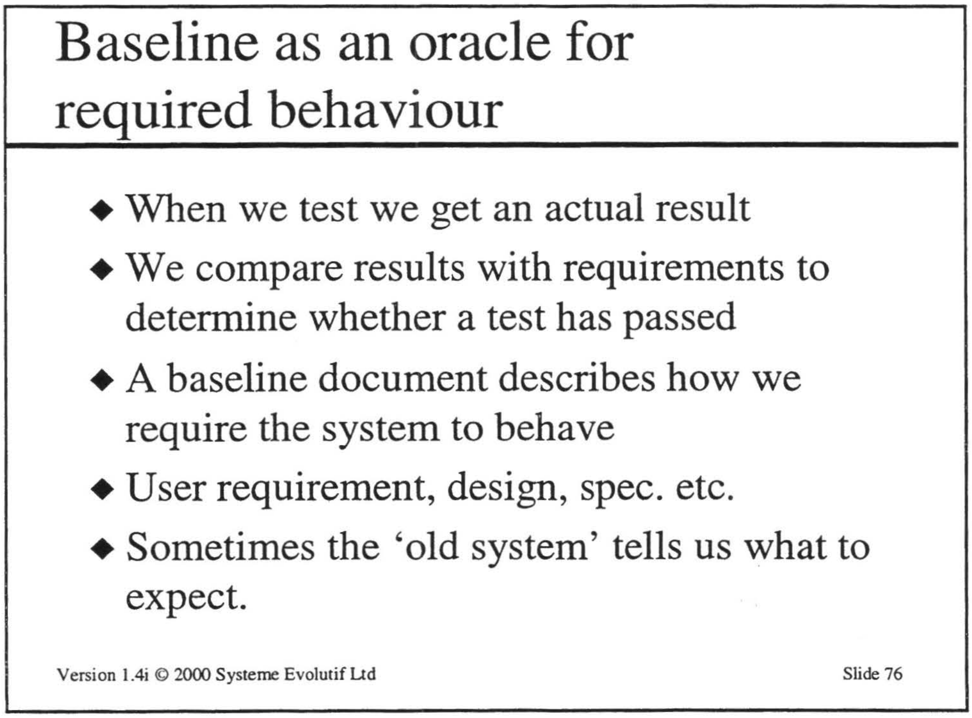
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 74



External specifications and baselines

- Specifications, requirements etc. define what the software is required to do
- Without requirements, developers cannot build, testers cannot test
  - – programmers need them to write the code
  - – testers need them to:
    - » identify the things that need testing
    - » compare test results with requirements
- Requirements, specs. Etc. are baselines.

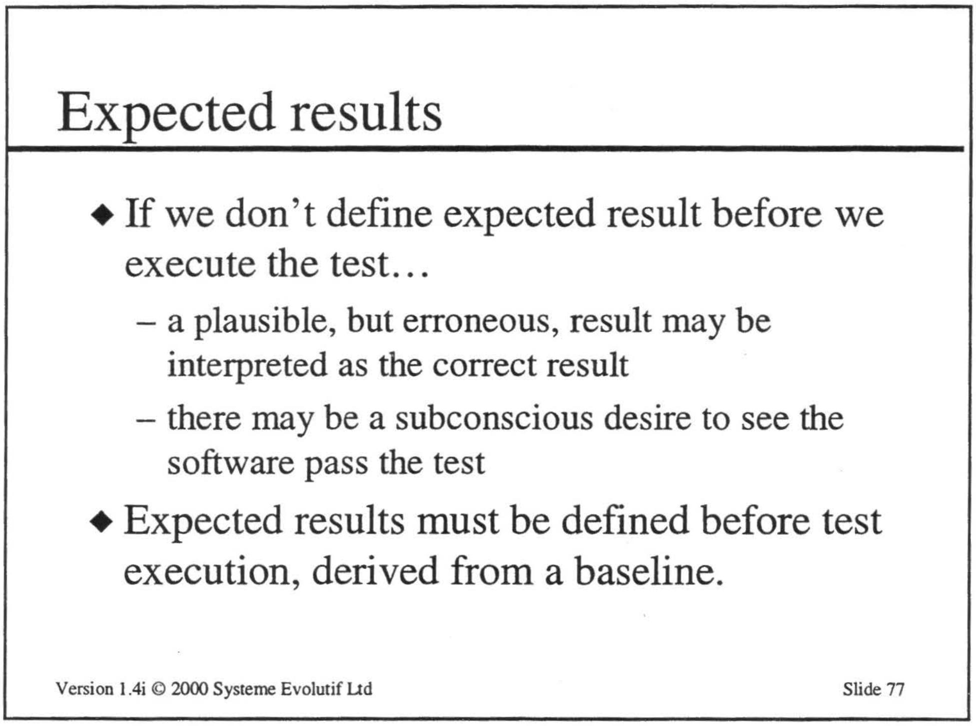
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 75



Baseline as an oracle for required behaviour

- When we test we get an actual result
- We compare results with requirements to determine whether a test has passed
- A baseline document describes how we require the system to behave
- User requirement, design, spec. etc.
- Sometimes the ‘old system’ tells us what to expect.

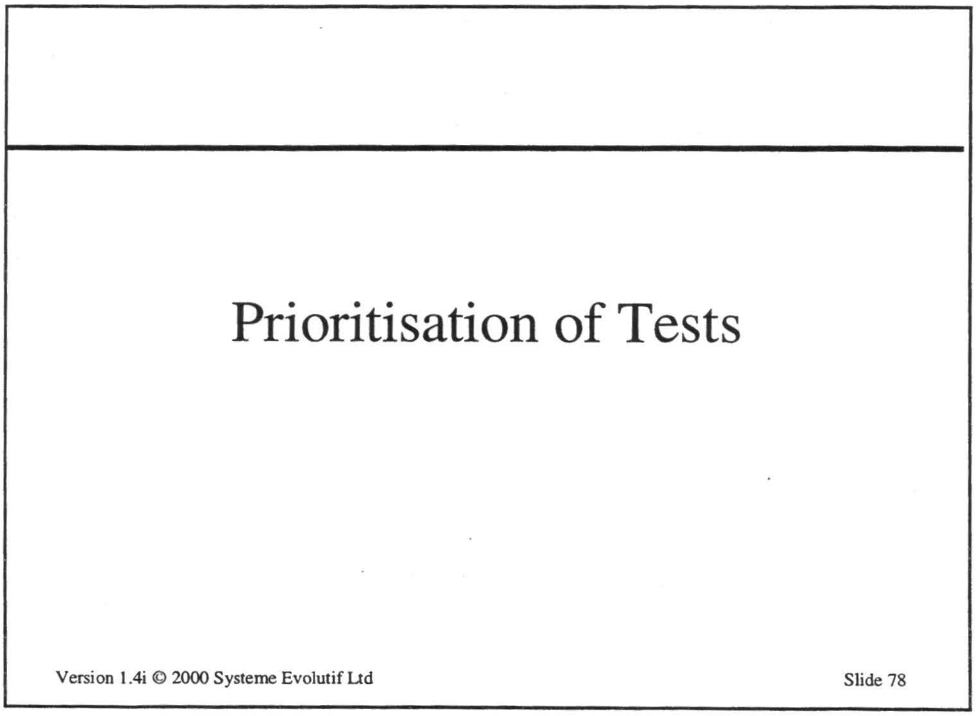
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 76



Expected results

- ◆ If we don't define expected result before we execute the test…
  - – a plausible, but erroneous, result may be interpreted as the correct result
  - – there may be a subconscious desire to see the software pass the test
- ◆ Expected results must be defined before test execution, derived from a baseline.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 77



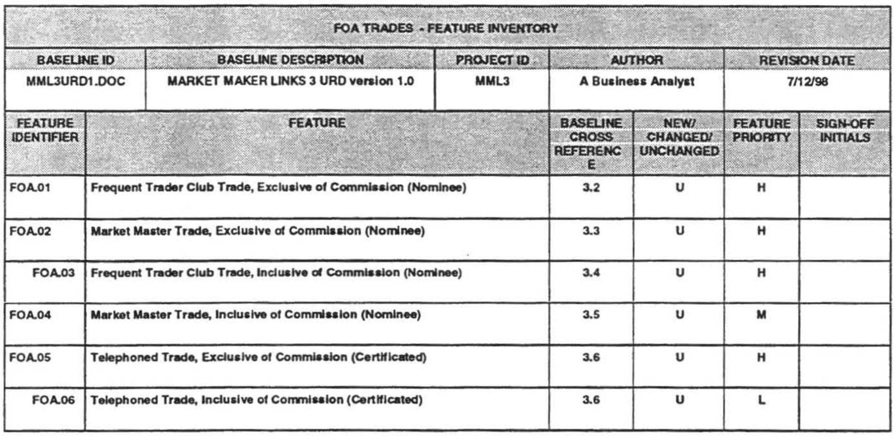
Prioritisation of Tests

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 78

Our coverage target

- There is no limit to how much testing we could do, so we *must* prioritise
- How much testing should we do?
  - what tests *could* we do?
  - which are the *most important*?
  - which can we *drop*?
- A test inventory derived provides us with a measurable target.

Feature inventory

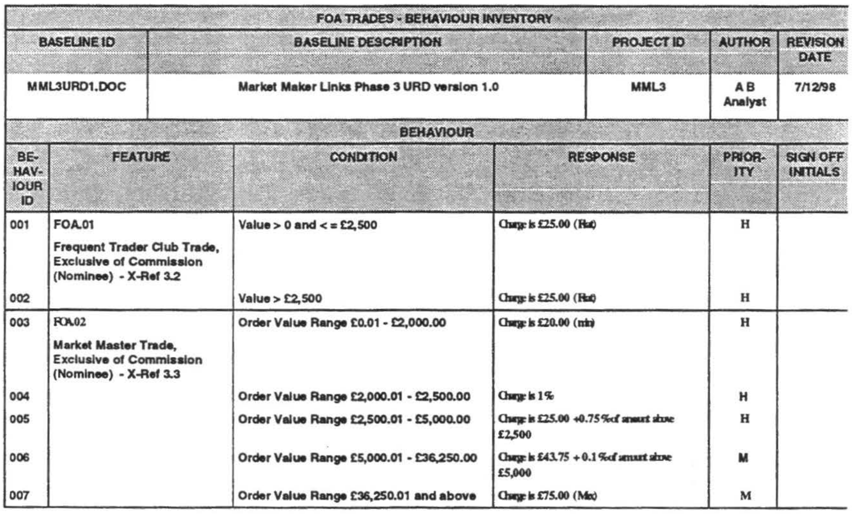


FOA TRADES - FEATURE INVENTORY

BASELINE ID	BASELINE DESCRIPTION	PROJECT ID	AUTHOR	REVISION DATE
MML3URD1.DOC	MARKET MAKER LINKS 3 URD version 1.0	MML3	A Business Analyst	7/12/98

FEATURE IDENTIFIER	FEATURE	BASELINE CROSS REFERENCE	NEW/ CHANGED/ UNCHANGED	FEATURE PRIORITY	SIGN-OFF INITIALS
FOA.01	Frequent Trader Club Trade, Exclusive of Commission (Nominee)	3.2	U	H	
FOA.02	Market Master Trade, Exclusive of Commission (Nominee)	3.3	U	H	
FOA.03	Frequent Trader Club Trade, Inclusive of Commission (Nominee)	3.4	U	H	
FOA.04	Market Master Trade, Inclusive of Commission (Nominee)	3.5	U	M	
FOA.05	Telephoned Trade, Exclusive of Commission (Certificated)	3.6	U	H	
FOA.06	Telephoned Trade, Inclusive of Commission (Certificated)	3.6	U	L	

Test condition inventory

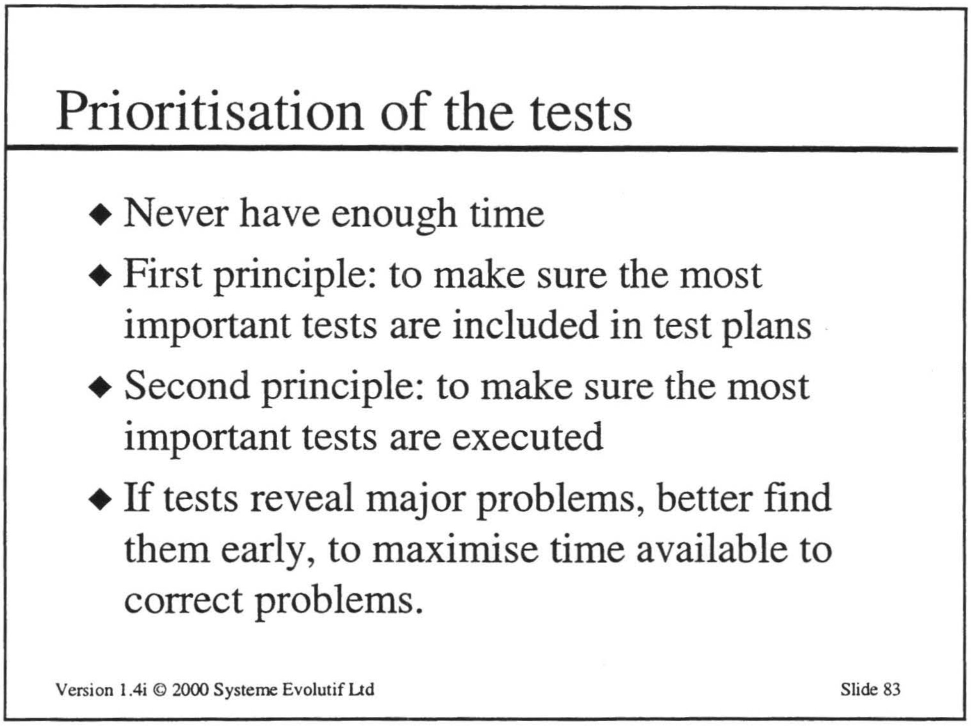


FOA TRADES - BEHAVIOUR INVENTORY				
BASELINE ID	BASELINE DESCRIPTION	PROJECT ID	AUTHOR	REVISION DATE
MML3URD1.DOC	Market Maker Links Phase 3 URD version 1.0	MML3	A B Analyst	7/12/98

BEHAVIOUR					
BE-HAV-IOUR ID	FEATURE	CONDITION	RESPONSE	PRIOR-ITY	SIGN OFF INITIALS
001	FOA.01 Frequent Trader Club Trade, Exclusive of Commission (Nominee) - X-Ref 3.2	Value > 0 and < = £2,500	Charge is £25.00 (Flat)	H	
002		Value > £2,500	Charge is £25.00 (Flat)	H	
003	FOA.02 Market Master Trade, Exclusive of Commission (Nominee) - X-Ref 3.3	Order Value Range £0.01 - £2,000.00	Charge is £20.00 (min)	H	
004		Order Value Range £2,000.01 - £2,500.00	Charge is 1%	H	
005		Order Value Range £2,500.01 - £5,000.00	Charge is £25.00 +0.75% of amount above £2,500	H	
006		Order Value Range £5,000.01 - £36,250.00	Charge is £43.75 + 0.1% of amount above £5,000	M	
007		Order Value Range £36,250.01 and above	Charge is £75.00 (Max)	M	

Test inventory and prioritisation

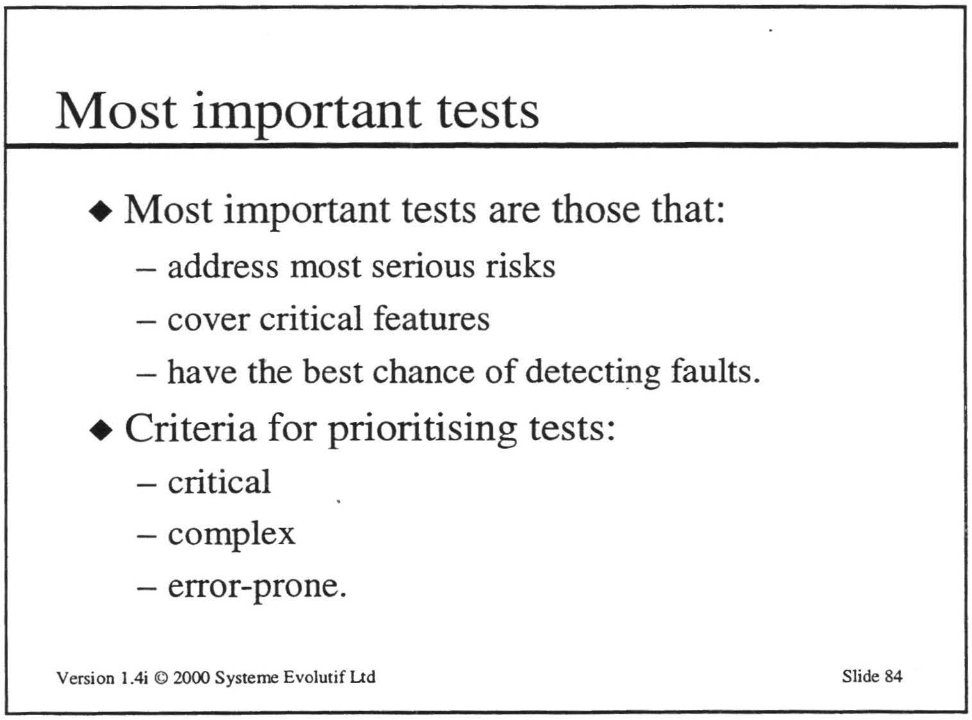
- To measure progress effectively, we must define the scope of the testing
- What we need to define:
  - the features of the system to be tested
  - the tests to be applied to give us confidence
- Inventory of tests enable us to:
  - prioritize the tests to stay within budget
  - estimate the effort required.



Prioritisation of the tests

- Never have enough time
- First principle: to make sure the most important tests are included in test plans
- Second principle: to make sure the most important tests are executed
- If tests reveal major problems, better find them early, to maximise time available to correct problems.

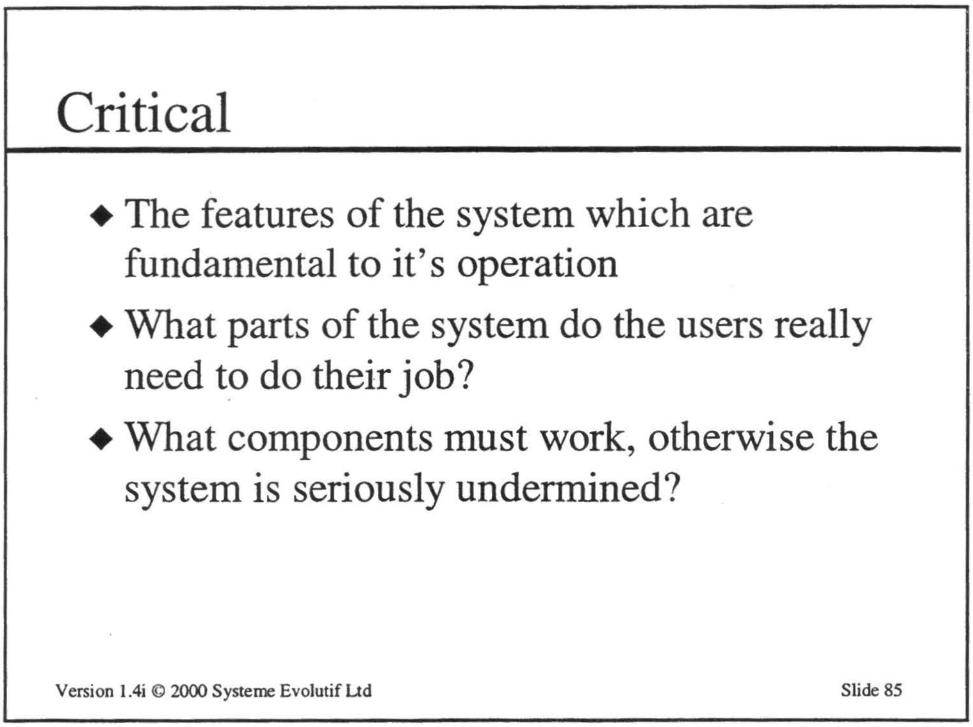
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 83



Most important tests

- Most important tests are those that:
  - address most serious risks
  - cover critical features
  - have the best chance of detecting faults.
- Criteria for prioritising tests:
  - critical
  - complex
  - error-prone.

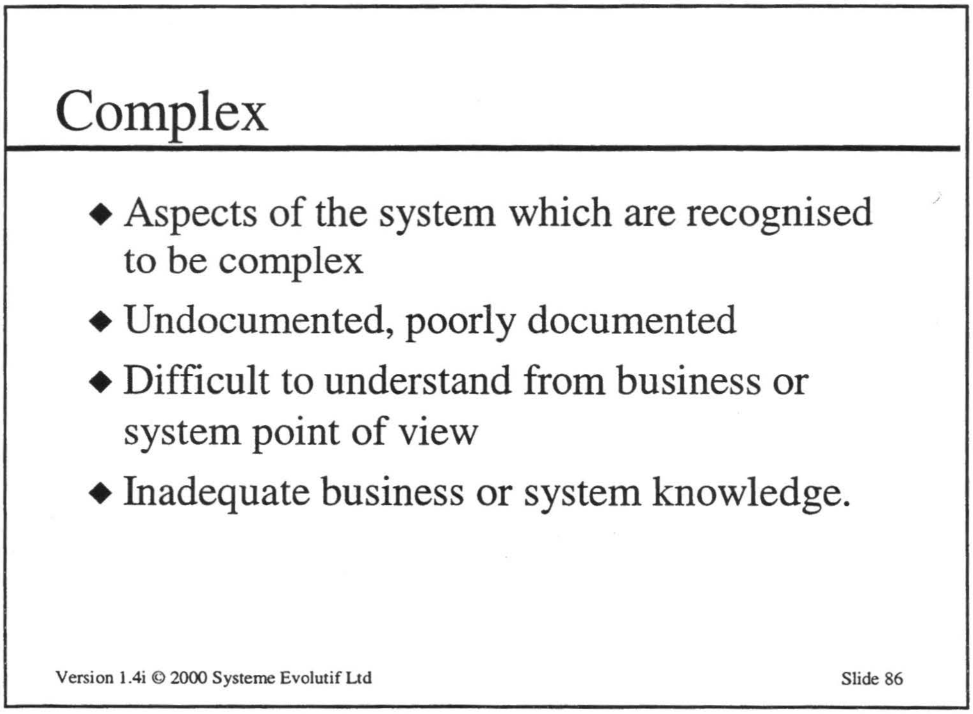
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 84



Critical

- The features of the system which are fundamental to it's operation
- What parts of the system do the users really need to do their job?
- What components must work, otherwise the system is seriously undermined?

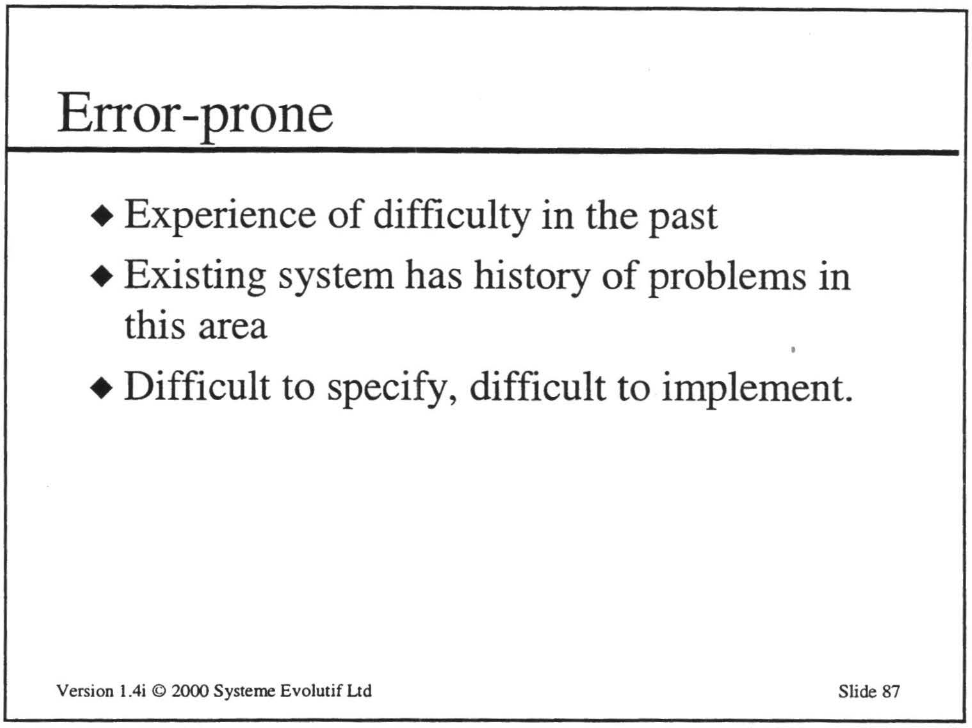
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 85



Complex

- Aspects of the system which are recognised to be complex
- Undocumented, poorly documented
- Difficult to understand from business or system point of view
- Inadequate business or system knowledge.

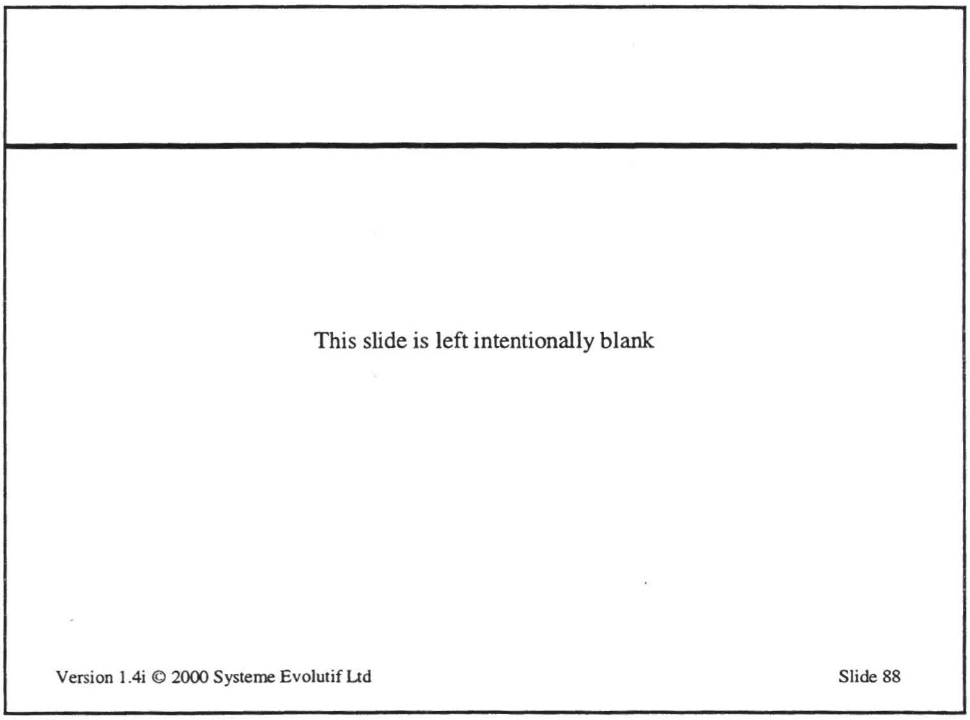
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 86



Error-prone

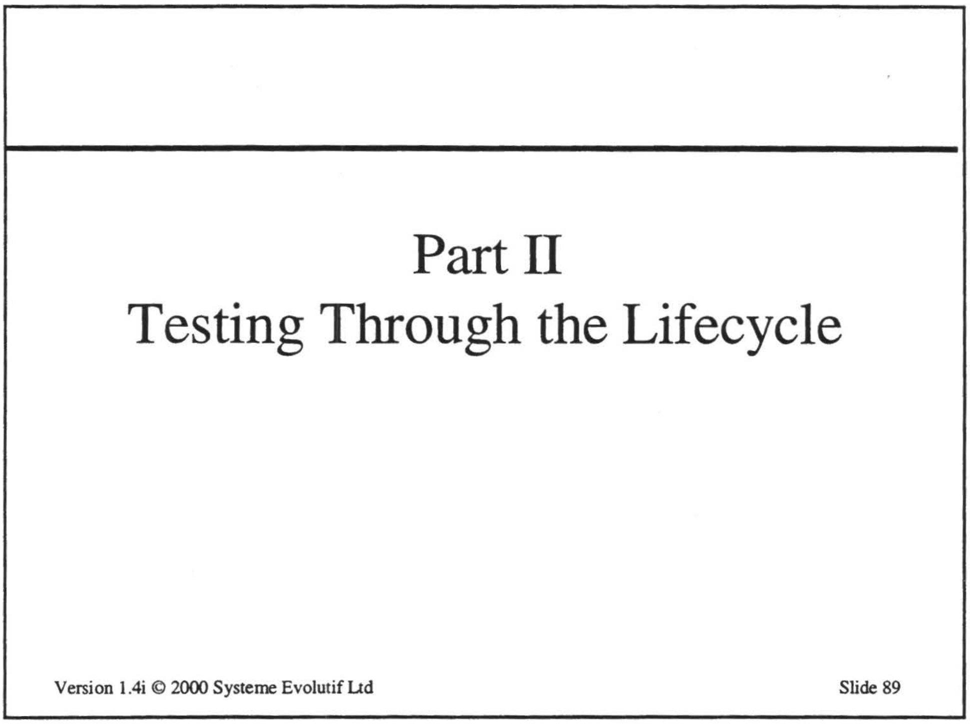
- ◆ Experience of difficulty in the past
- ◆ Existing system has history of problems in this area
- ◆ Difficult to specify, difficult to implement.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 87



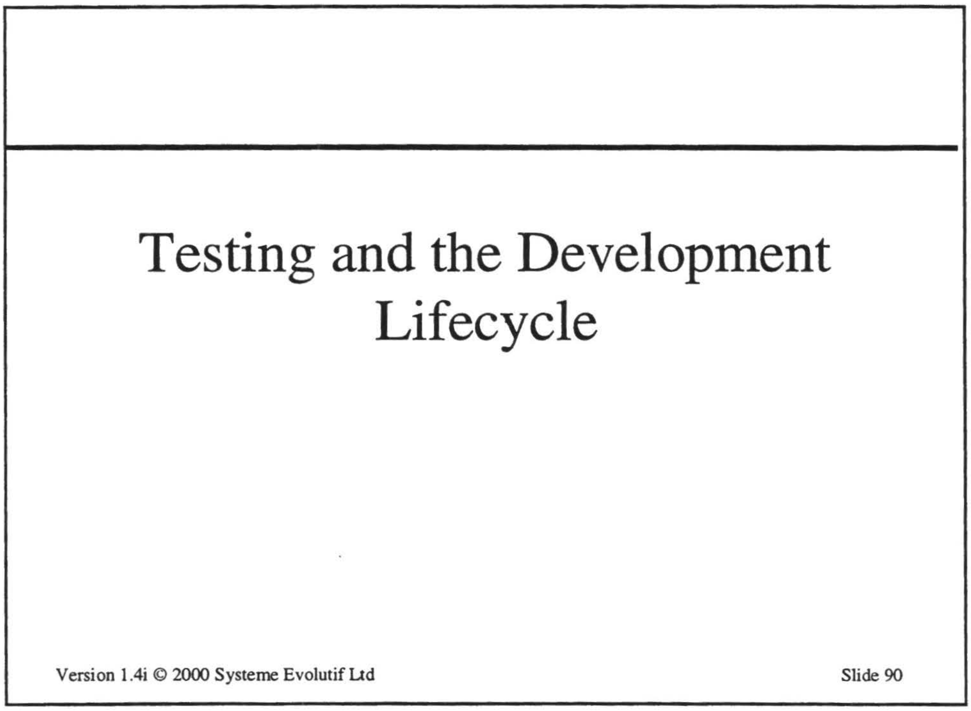
This slide is left intentionally blank

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 88



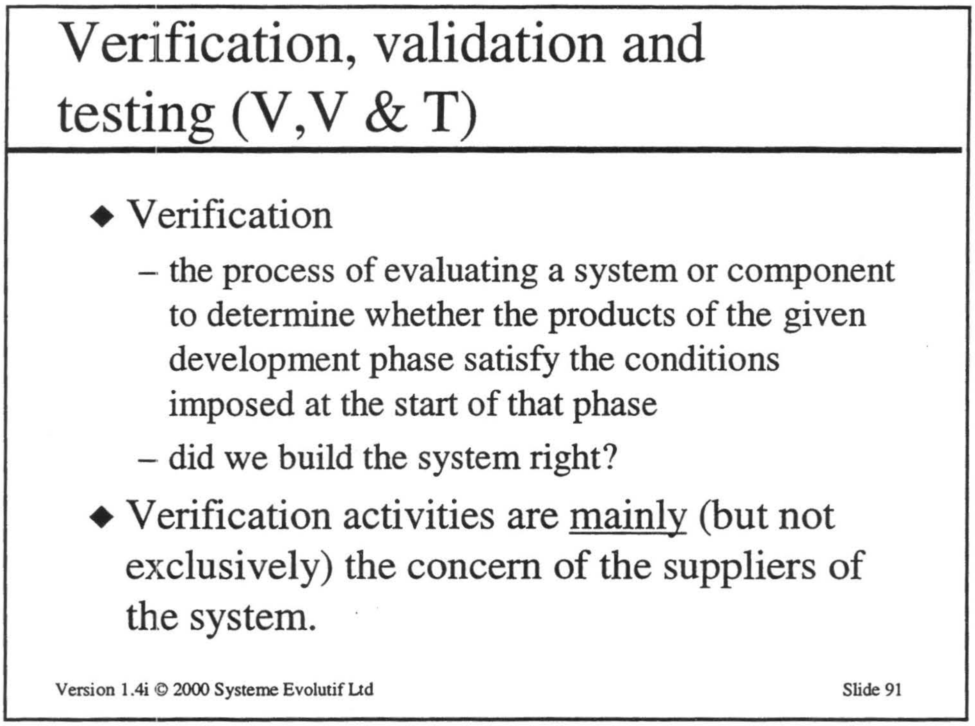
Part II
Testing Through the Lifecycle

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 89



Testing and the Development Lifecycle

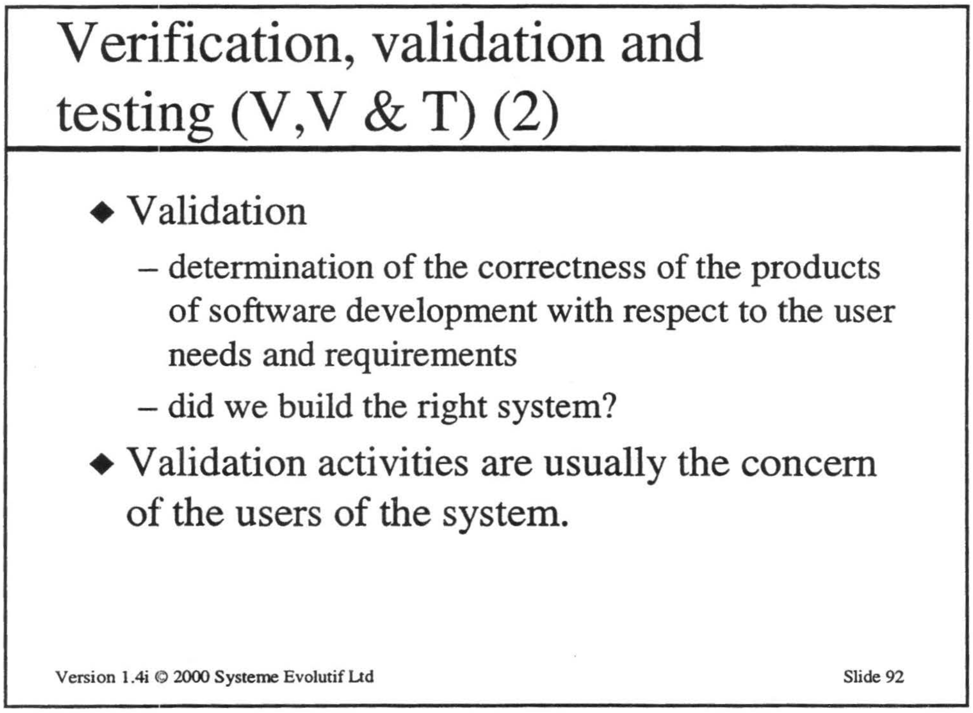
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 90



Verification, validation and testing (V,V & T)

- Verification
  - the process of evaluating a system or component to determine whether the products of the given development phase satisfy the conditions imposed at the start of that phase
  - did we build the system right?
- Verification activities are <u>mainly</u> (but not exclusively) the concern of the suppliers of the system.

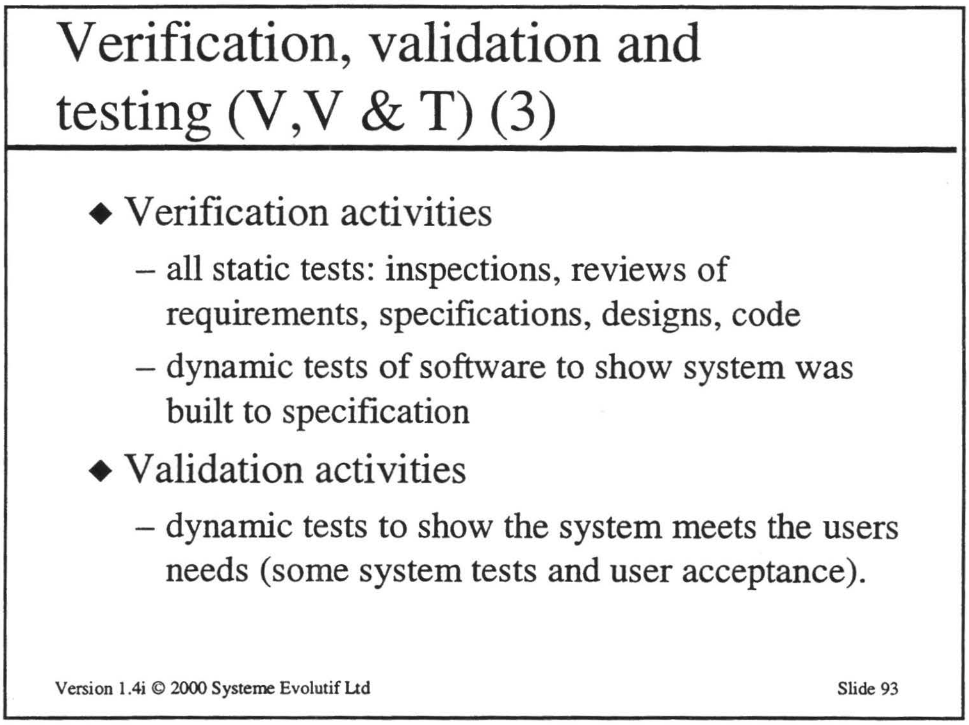
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 91



Verification, validation and testing (V,V & T) (2)

- Validation
  - determination of the correctness of the products of software development with respect to the user needs and requirements
  - did we build the right system?
- Validation activities are usually the concern of the users of the system.

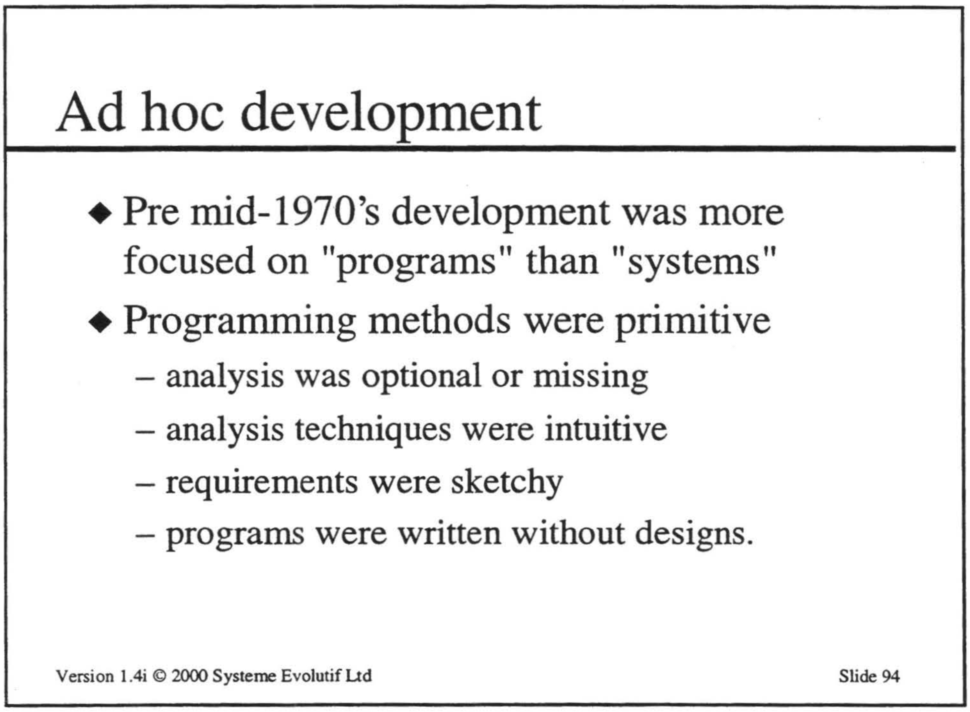
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 92



Verification, validation and testing (V,V & T) (3)

- ◆ Verification activities
  - – all static tests: inspections, reviews of requirements, specifications, designs, code
  - – dynamic tests of software to show system was built to specification
- ◆ Validation activities
  - – dynamic tests to show the system meets the users needs (some system tests and user acceptance).

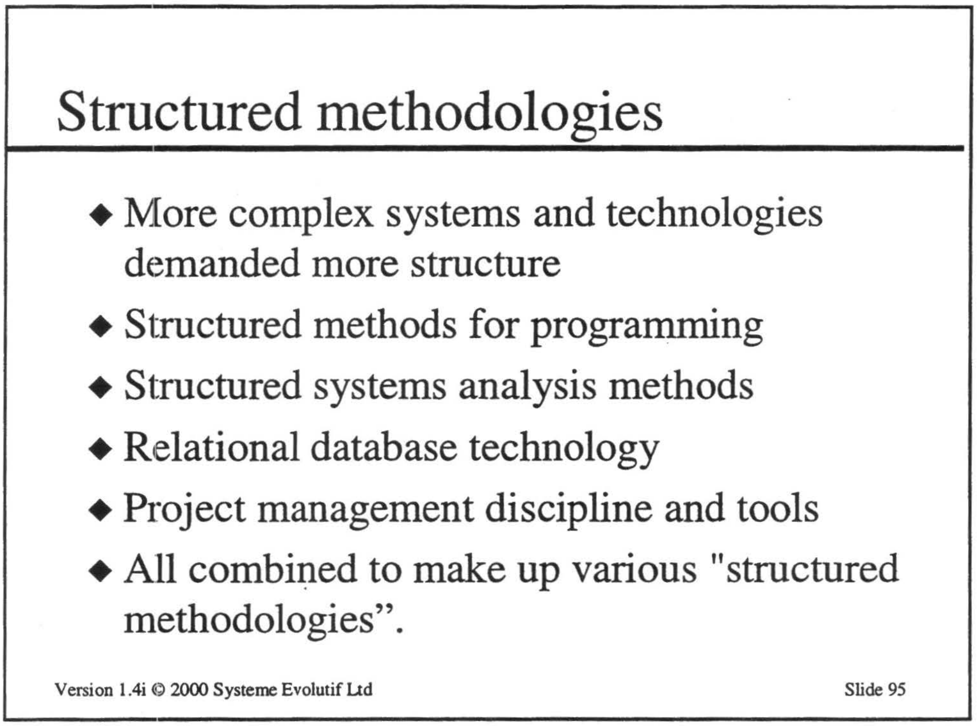
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 93



Ad hoc development

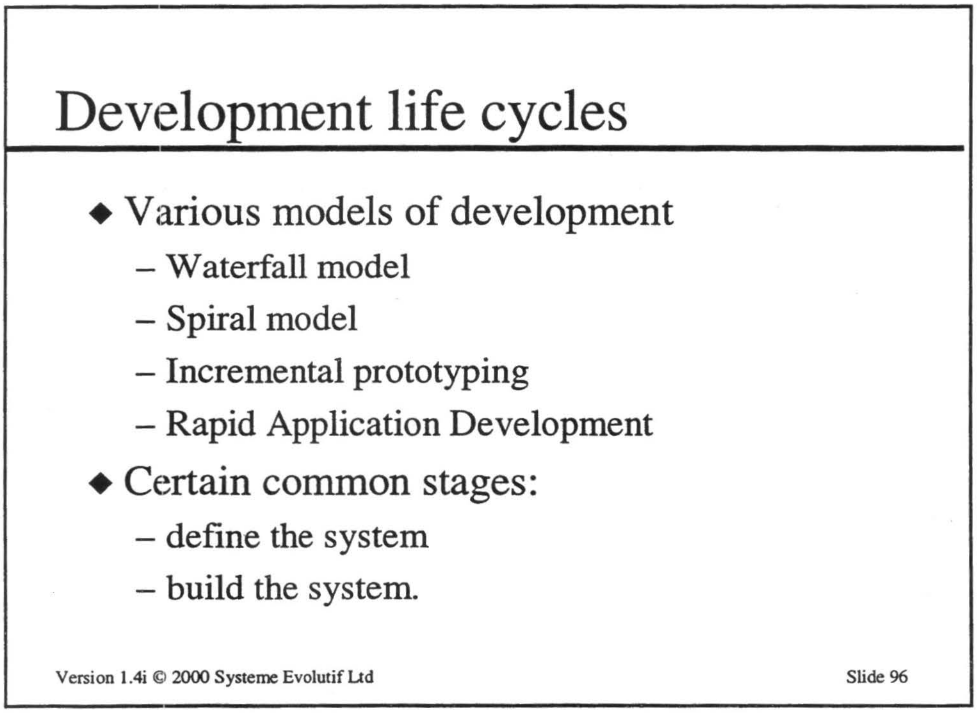
- ◆ Pre mid-1970's development was more focused on "programs" than "systems"
- ◆ Programming methods were primitive
  - – analysis was optional or missing
  - – analysis techniques were intuitive
  - – requirements were sketchy
  - – programs were written without designs.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 94



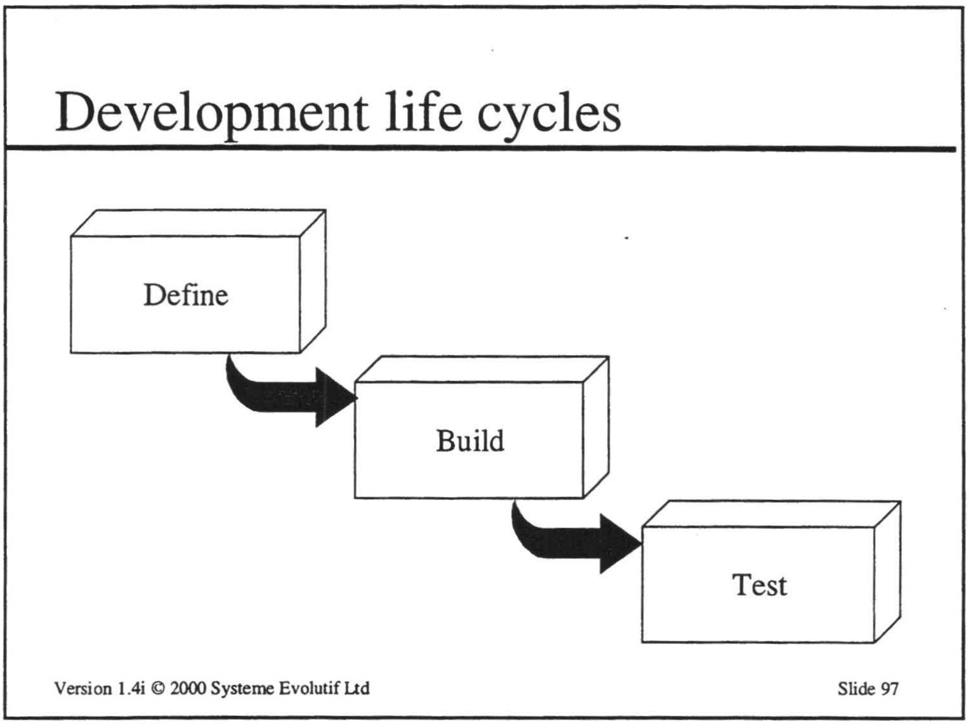
Structured methodologies

- More complex systems and technologies demanded more structure
- Structured methods for programming
- Structured systems analysis methods
- Relational database technology
- Project management discipline and tools
- All combined to make up various "structured methodologies”.



Development life cycles

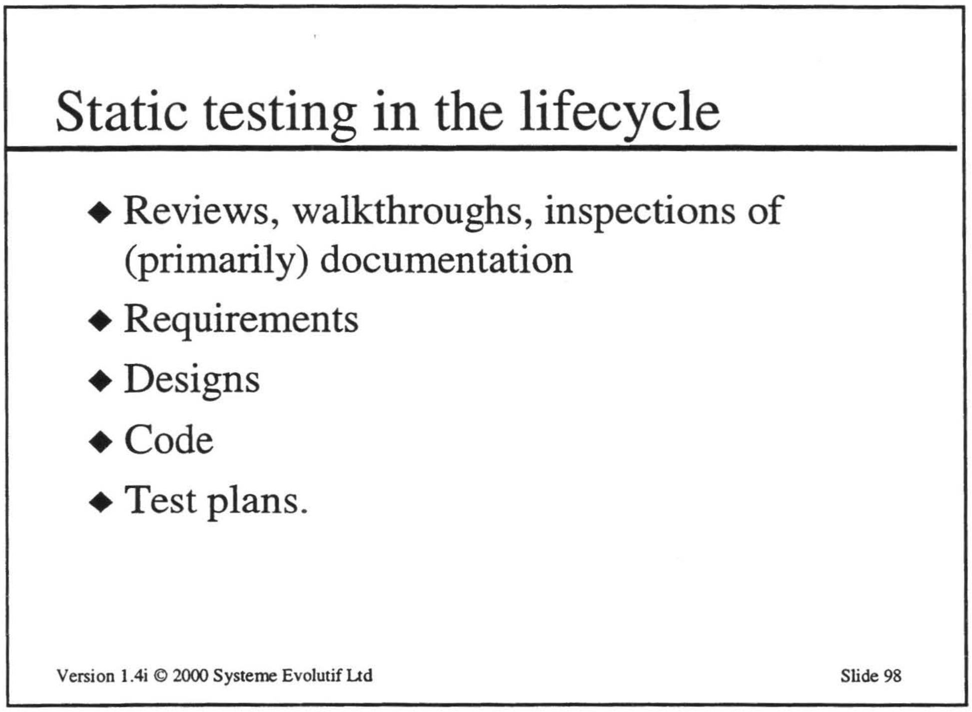
- Various models of development
  - Waterfall model
  - Spiral model
  - Incremental prototyping
  - Rapid Application Development
- Certain common stages:
  - define the system
  - build the system.



Development life cycles

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 97

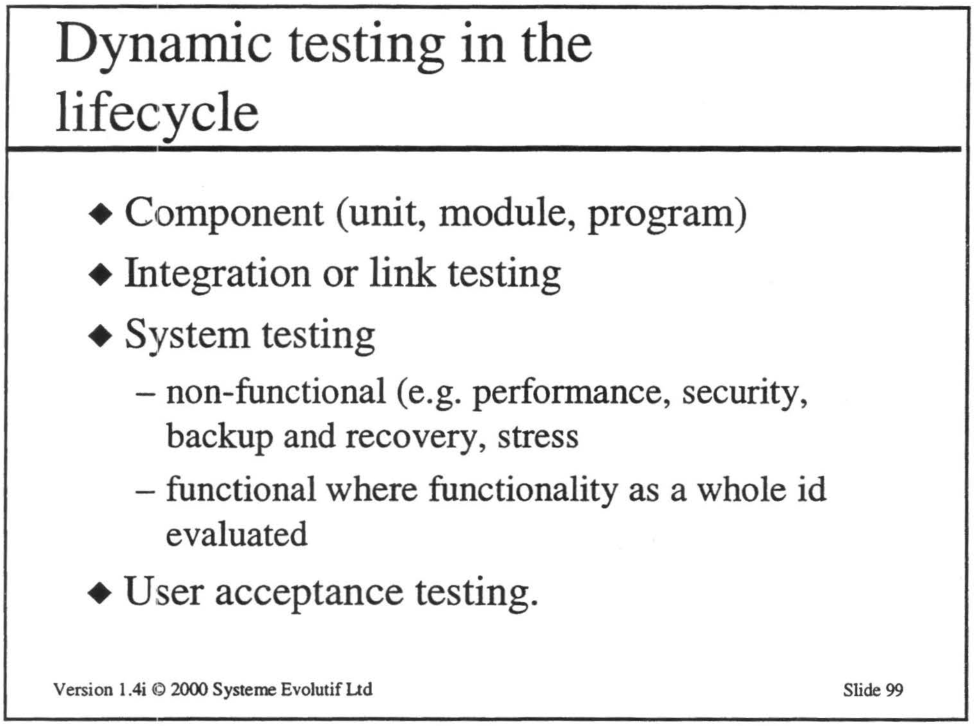


Static testing in the lifecycle

- Reviews, walkthroughs, inspections of (primarily) documentation
- Requirements
- Designs
- Code
- Test plans.

Version 1.4i © 2000 Systeme Evolutif Ltd

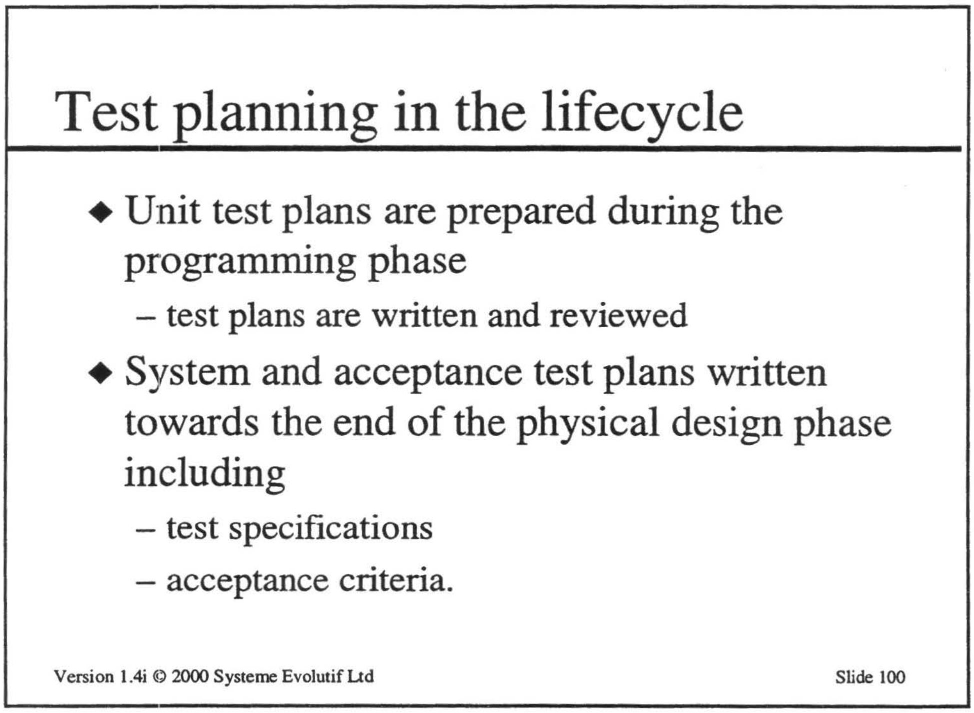
Slide 98



Dynamic testing in the lifecycle

- ◆ Component (unit, module, program)
- ◆ Integration or link testing
- ◆ System testing
  - – non-functional (e.g. performance, security, backup and recovery, stress
  - – functional where functionality as a whole id evaluated
- ◆ User acceptance testing.

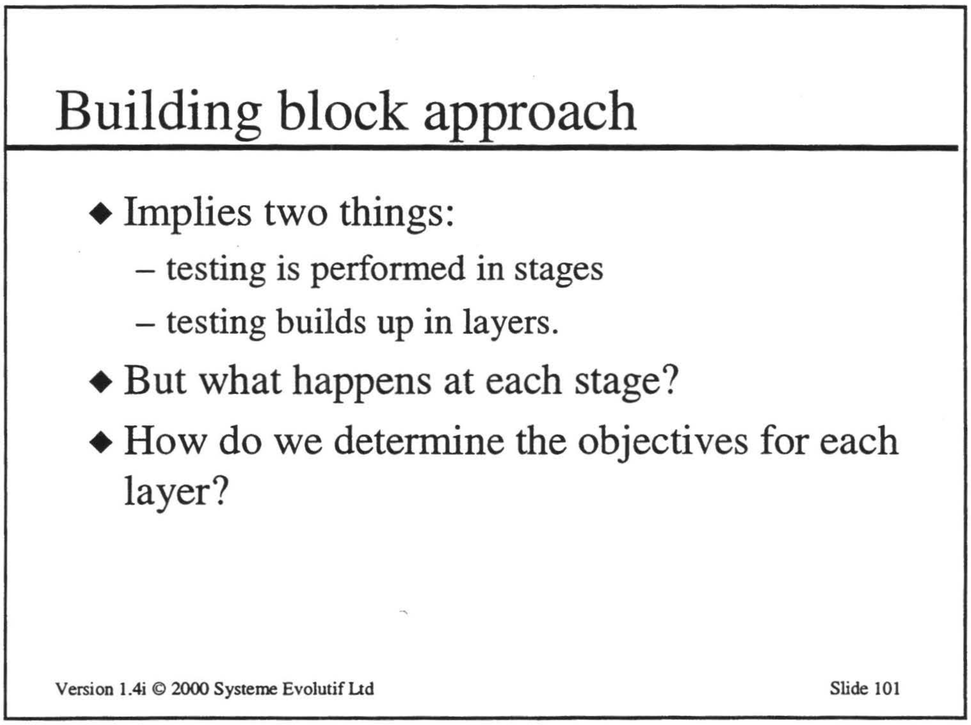
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 99



Test planning in the lifecycle

- ◆ Unit test plans are prepared during the programming phase
  - – test plans are written and reviewed
- ◆ System and acceptance test plans written towards the end of the physical design phase including
  - – test specifications
  - – acceptance criteria.

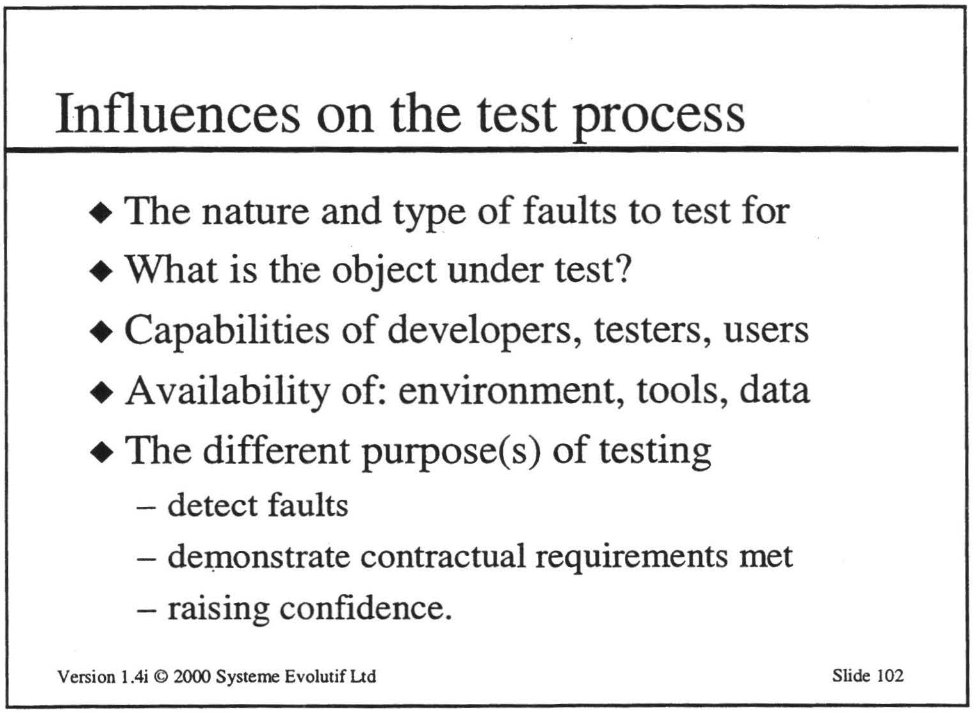
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 100



Building block approach

- ◆ Implies two things:
  - – testing is performed in stages
  - – testing builds up in layers.
- ◆ But what happens at each stage?
- ◆ How do we determine the objectives for each layer?

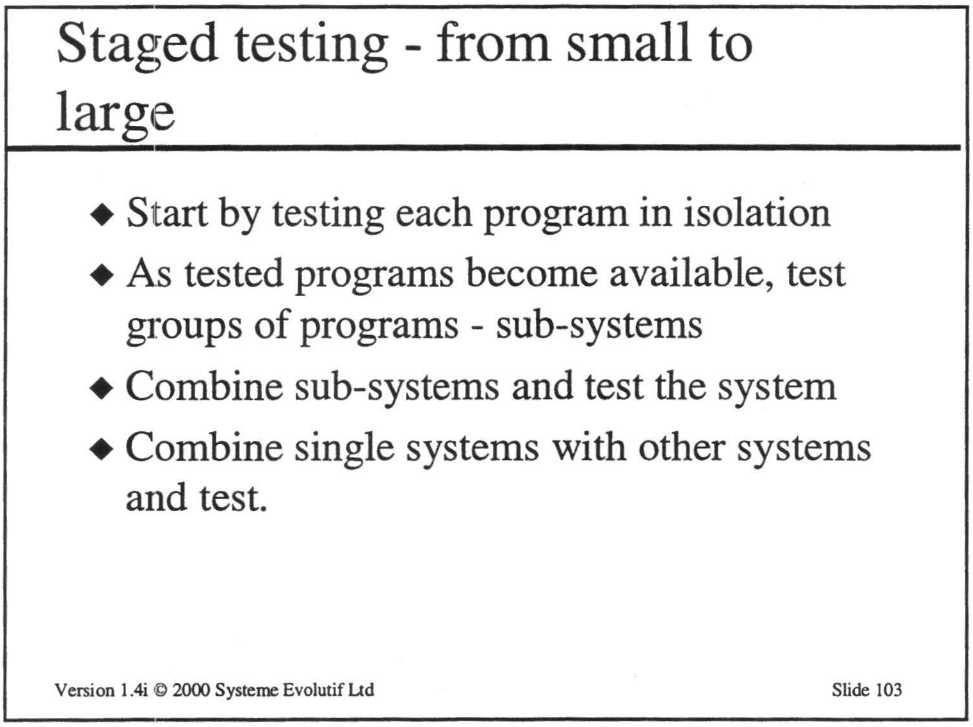
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 101



Influences on the test process

- ◆ The nature and type of faults to test for
- ◆ What is the object under test?
- ◆ Capabilities of developers, testers, users
- ◆ Availability of: environment, tools, data
- ◆ The different purpose(s) of testing
  - – detect faults
  - – demonstrate contractual requirements met
  - – raising confidence.

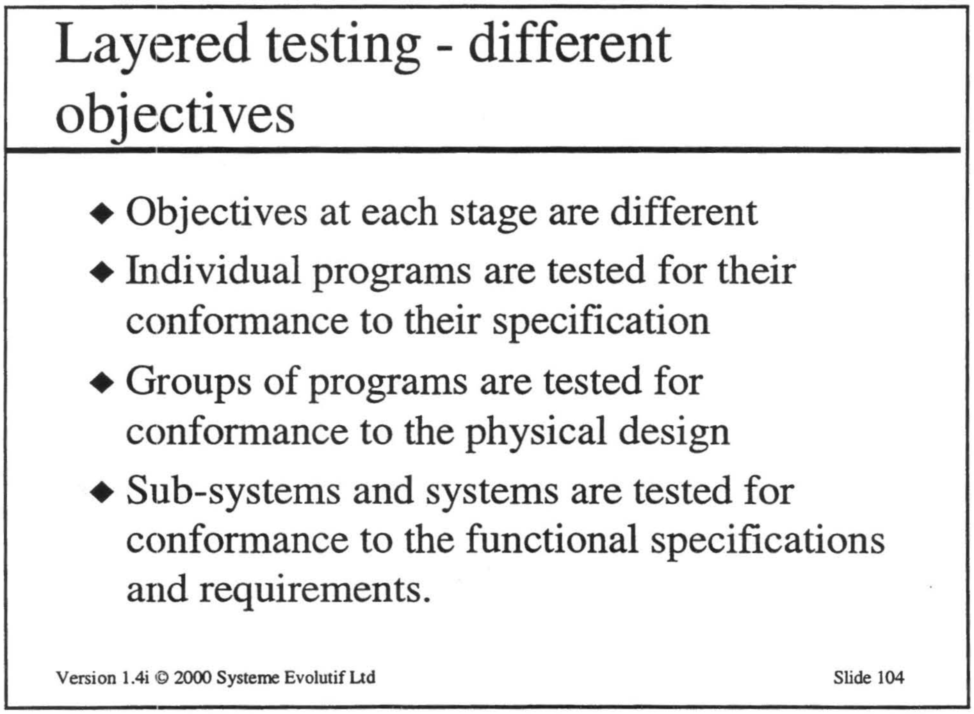
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 102



Staged testing - from small to large

- Start by testing each program in isolation
- As tested programs become available, test groups of programs - sub-systems
- Combine sub-systems and test the system
- Combine single systems with other systems and test.

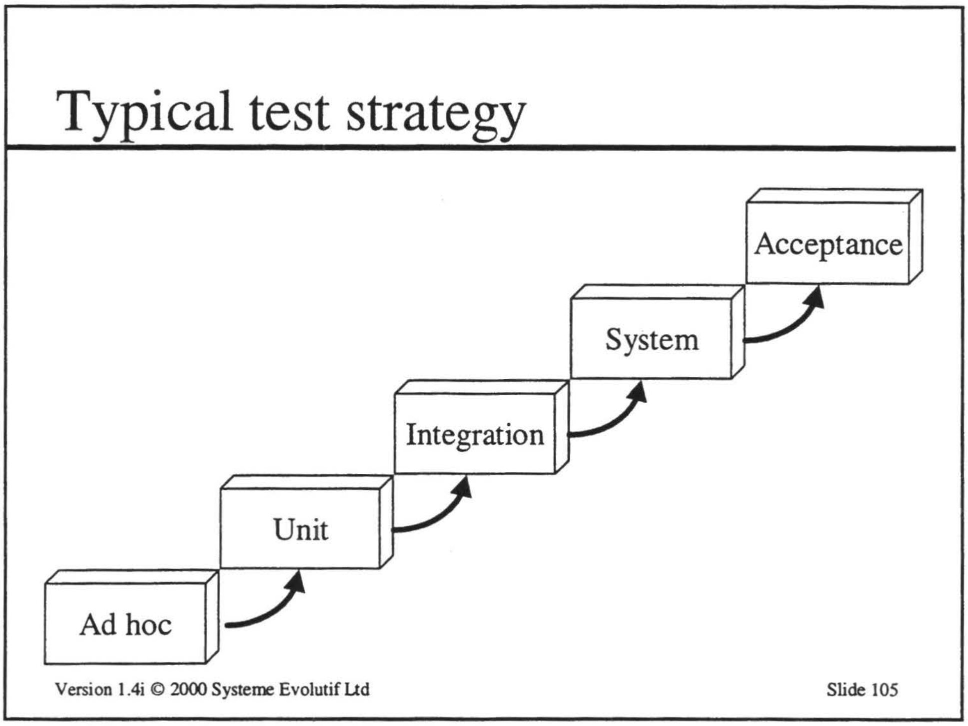
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 103



Layered testing - different objectives

- Objectives at each stage are different
- Individual programs are tested for their conformance to their specification
- Groups of programs are tested for conformance to the physical design
- Sub-systems and systems are tested for conformance to the functional specifications and requirements.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 104



Typical test strategy

Acceptance

System

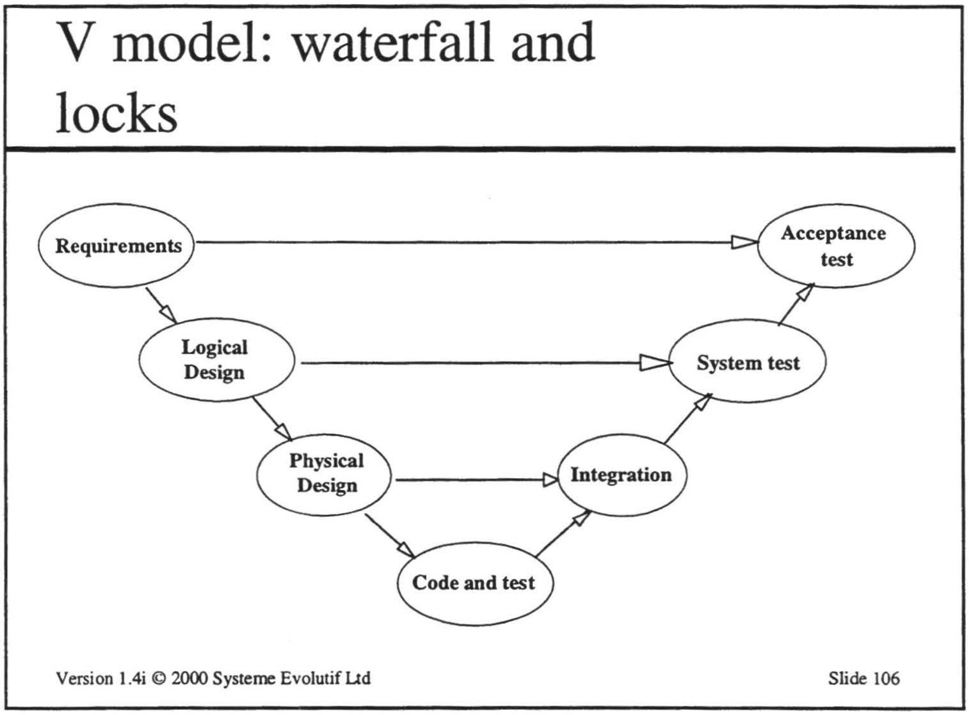
Integration

Unit

Ad hoc

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 105



V model: waterfall and locks

Requirements

Acceptance test

Logical Design

System test

Physical Design

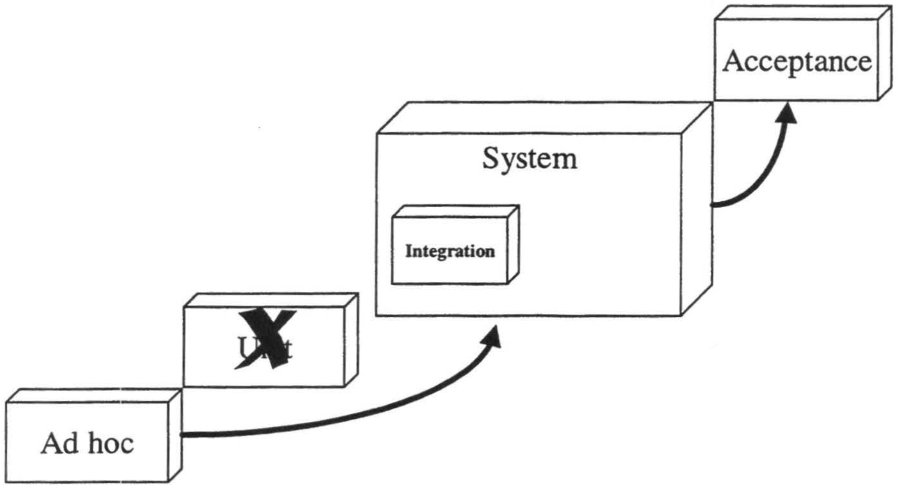
Integration

Code and test

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 106

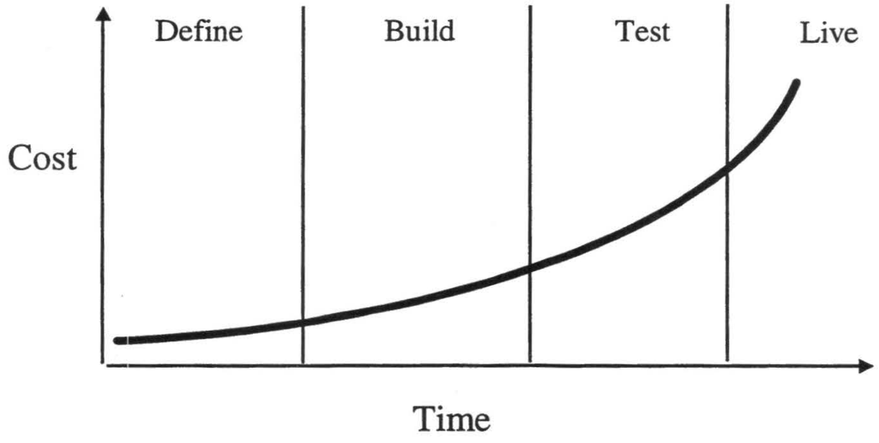
Typical test practice



Version 1.4i © 2000 Systeme Evolutif Ltd

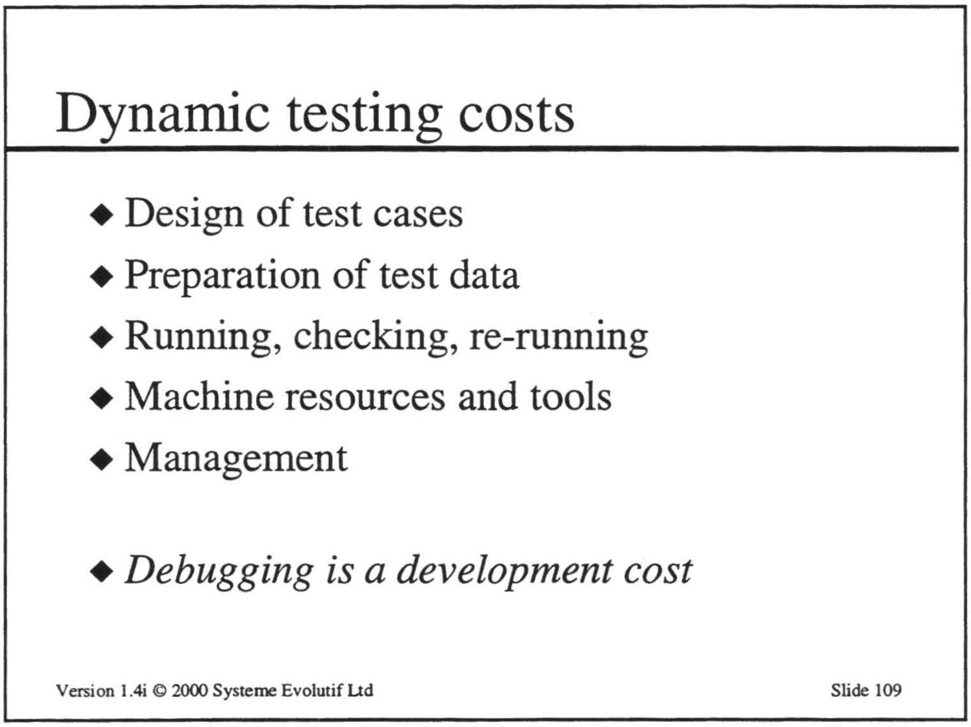
Slide 107

Fault cost curve



Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 108

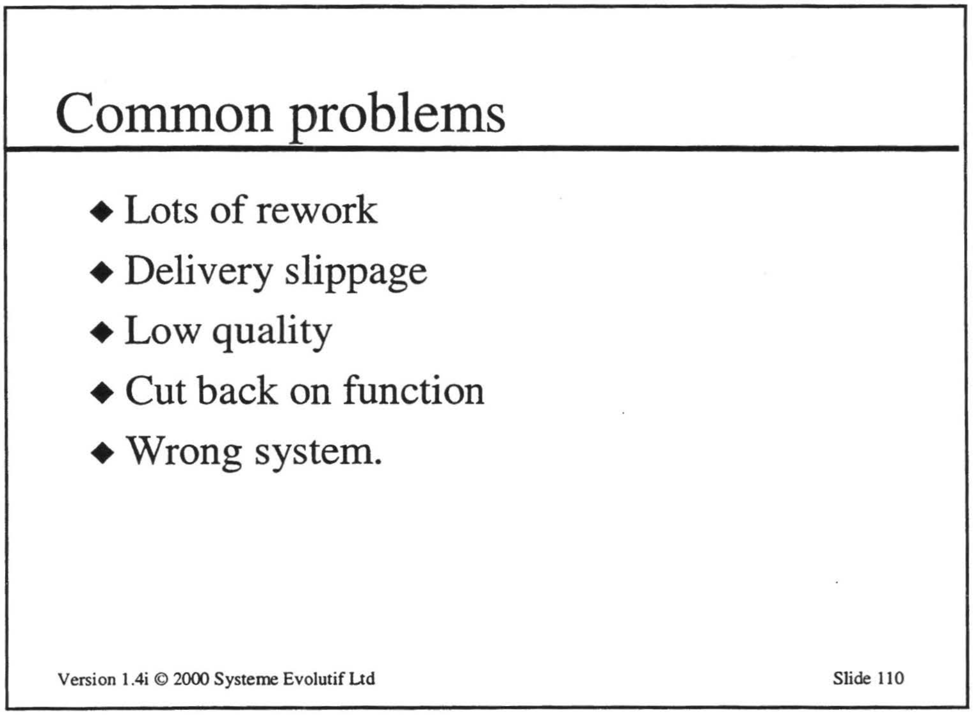


Dynamic testing costs

- Design of test cases
- Preparation of test data
- Running, checking, re-running
- Machine resources and tools
- Management

- *Debugging is a development cost*

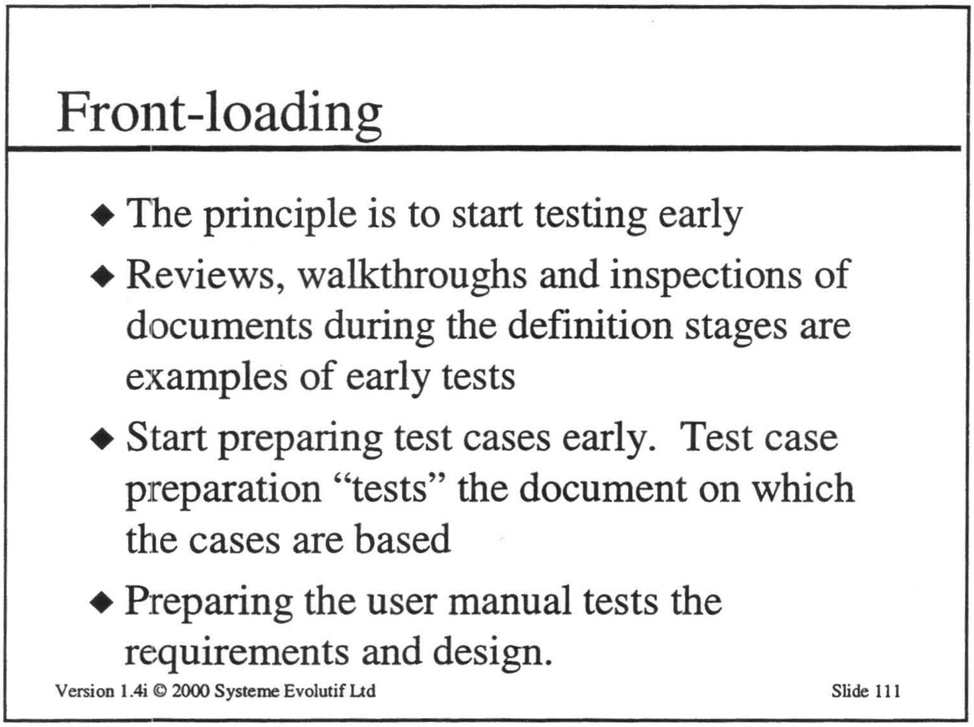
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 109



Common problems

- Lots of rework
- Delivery slippage
- Low quality
- Cut back on function
- Wrong system.

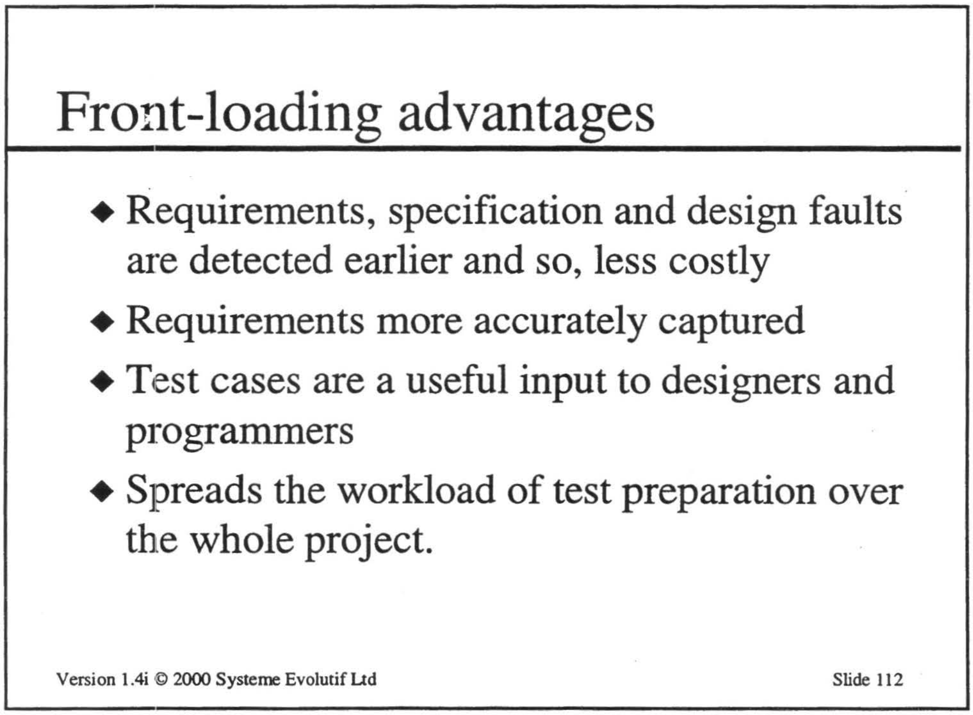
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 110



Front-loading

- The principle is to start testing early
- Reviews, walkthroughs and inspections of documents during the definition stages are examples of early tests
- Start preparing test cases early. Test case preparation "tests" the document on which the cases are based
- Preparing the user manual tests the requirements and design.

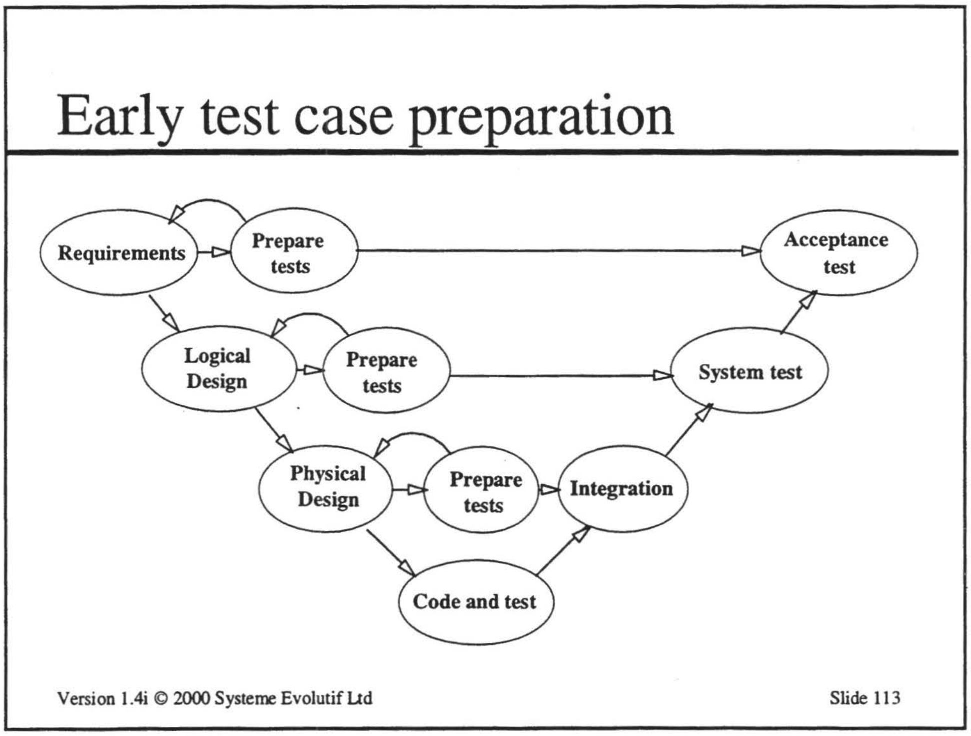
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 111



Front-loading advantages

- Requirements, specification and design faults are detected earlier and so, less costly
- Requirements more accurately captured
- Test cases are a useful input to designers and programmers
- Spreads the workload of test preparation over the whole project.

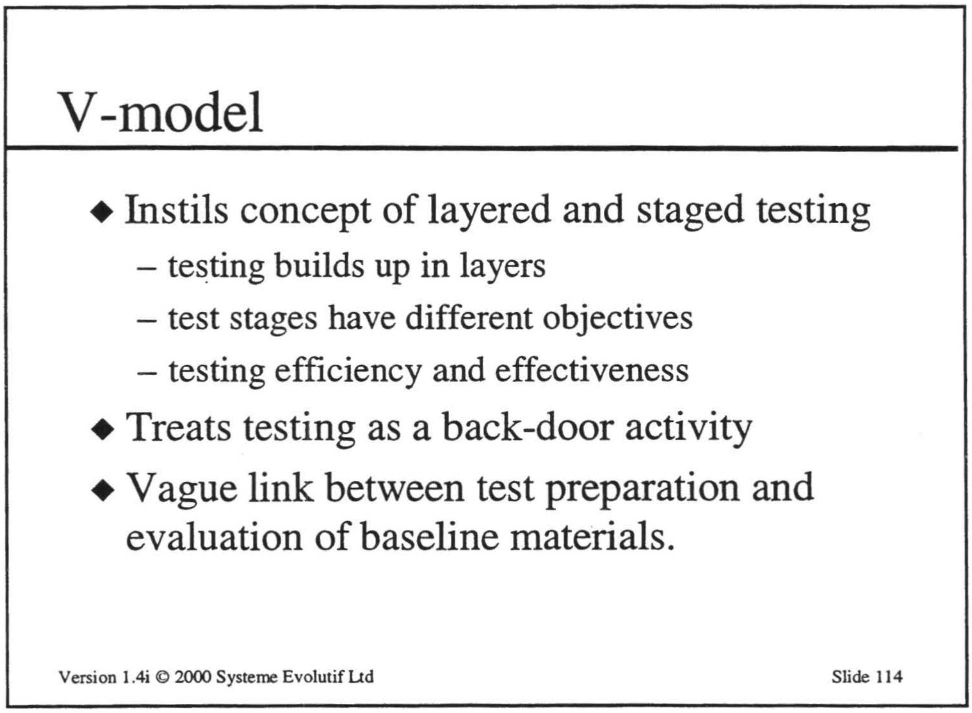
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 112



Early test case preparation

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 113

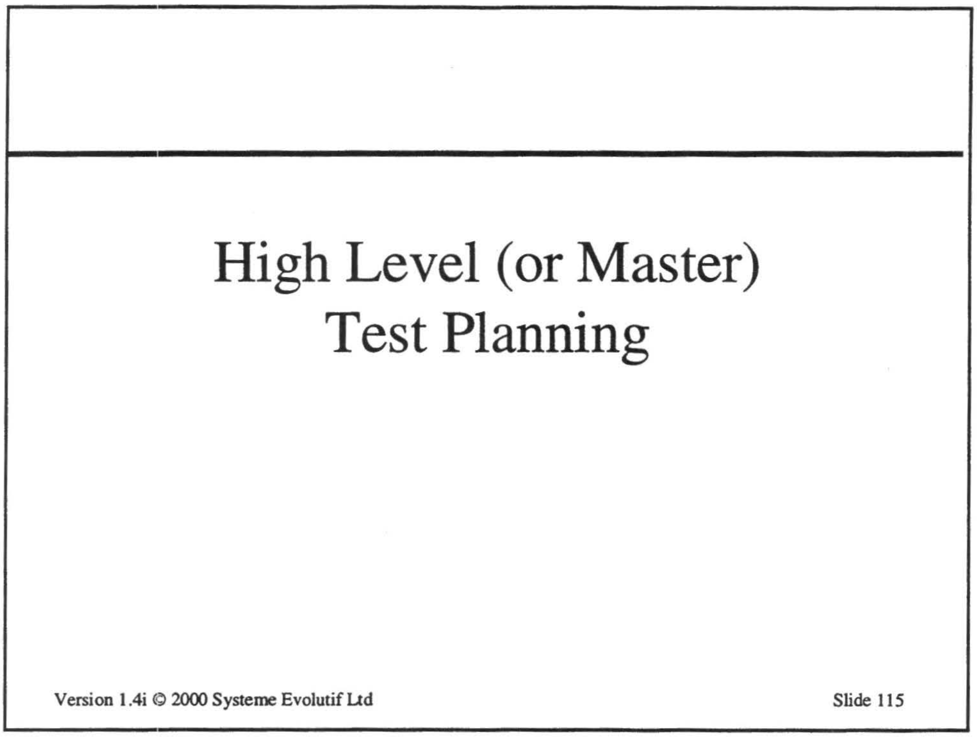


V-model

- Instils concept of layered and staged testing
  - testing builds up in layers
  - test stages have different objectives
  - testing efficiency and effectiveness
- Treats testing as a back-door activity
- Vague link between test preparation and evaluation of baseline materials.

Version 1.4i © 2000 Systeme Evolutif Ltd

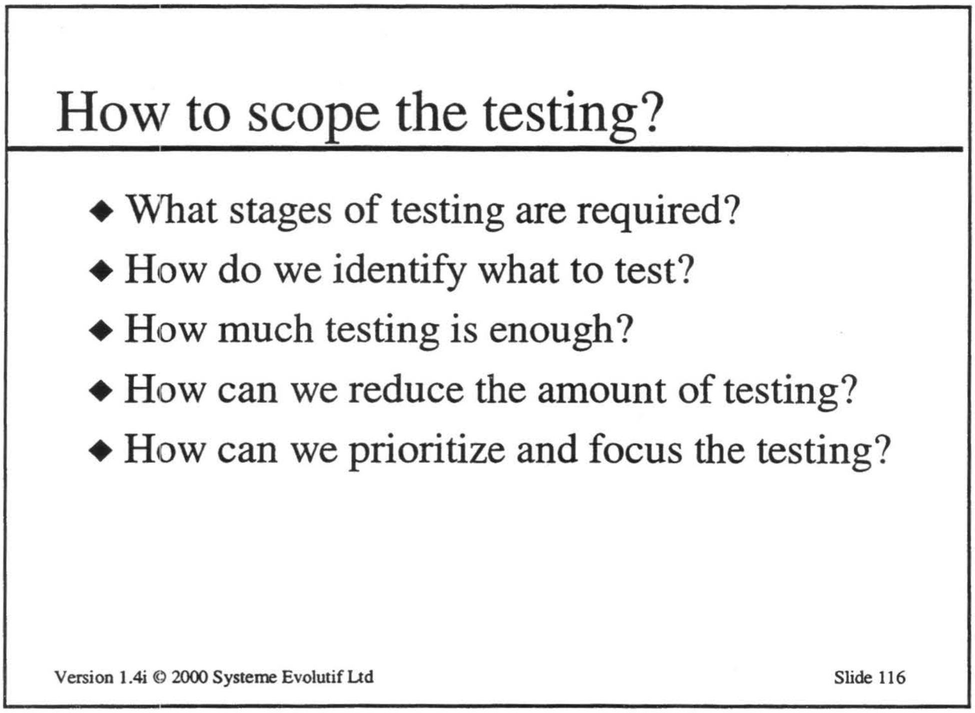
Slide 114



High Level (or Master) Test Planning

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 115

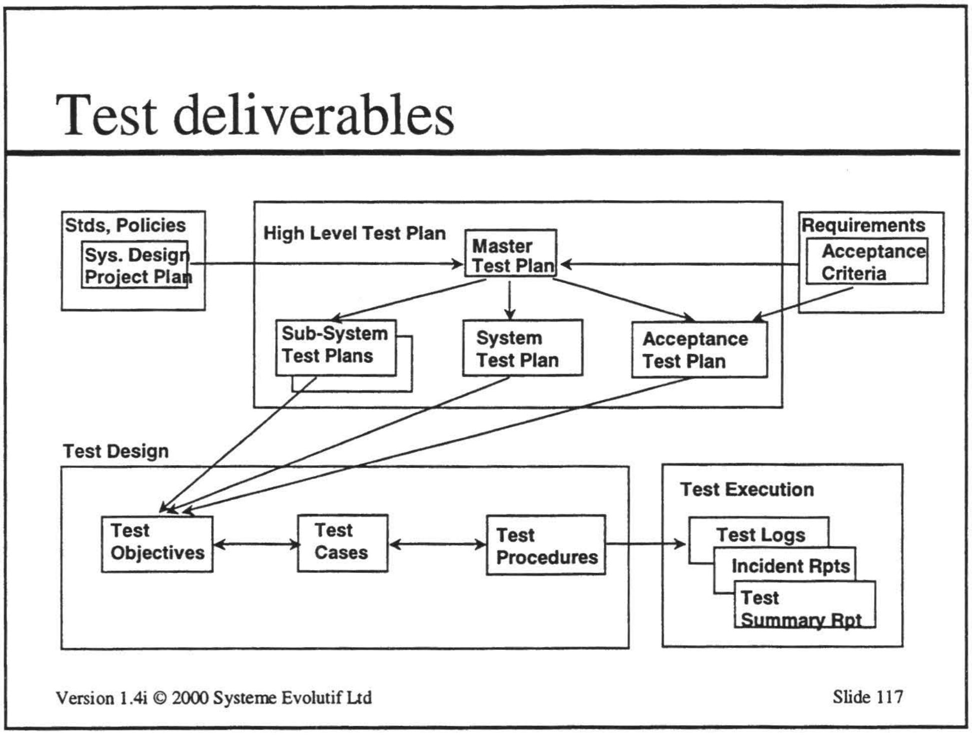


How to scope the testing?

- What stages of testing are required?
- How do we identify what to test?
- How much testing is enough?
- How can we reduce the amount of testing?
- How can we prioritize and focus the testing?

Version 1.4i © 2000 Systeme Evolutif Ltd

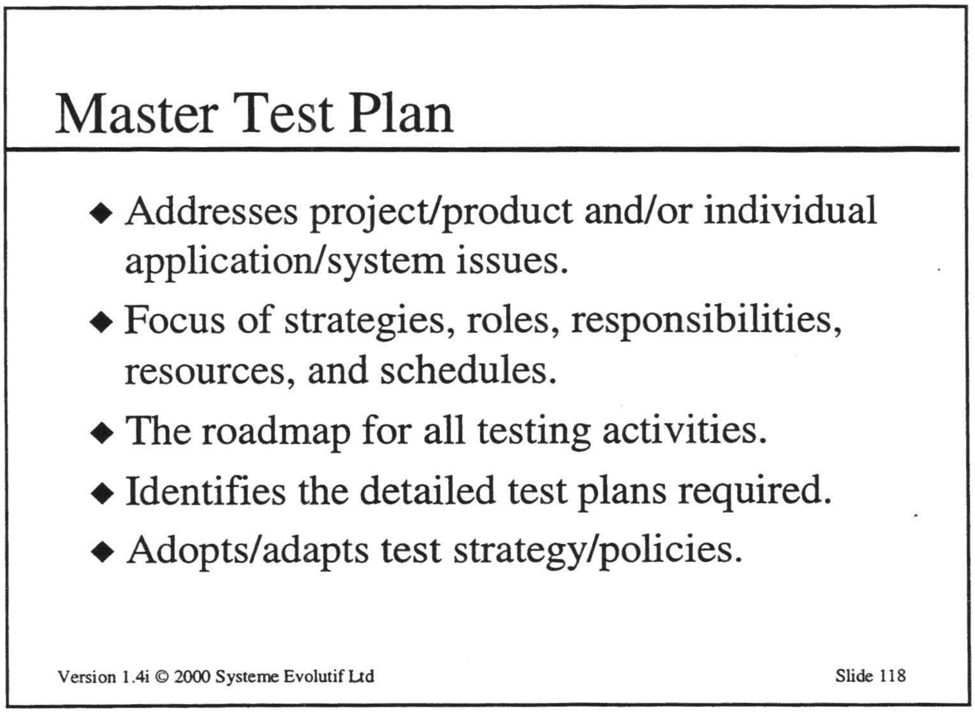
Slide 116



Test deliverables

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 117

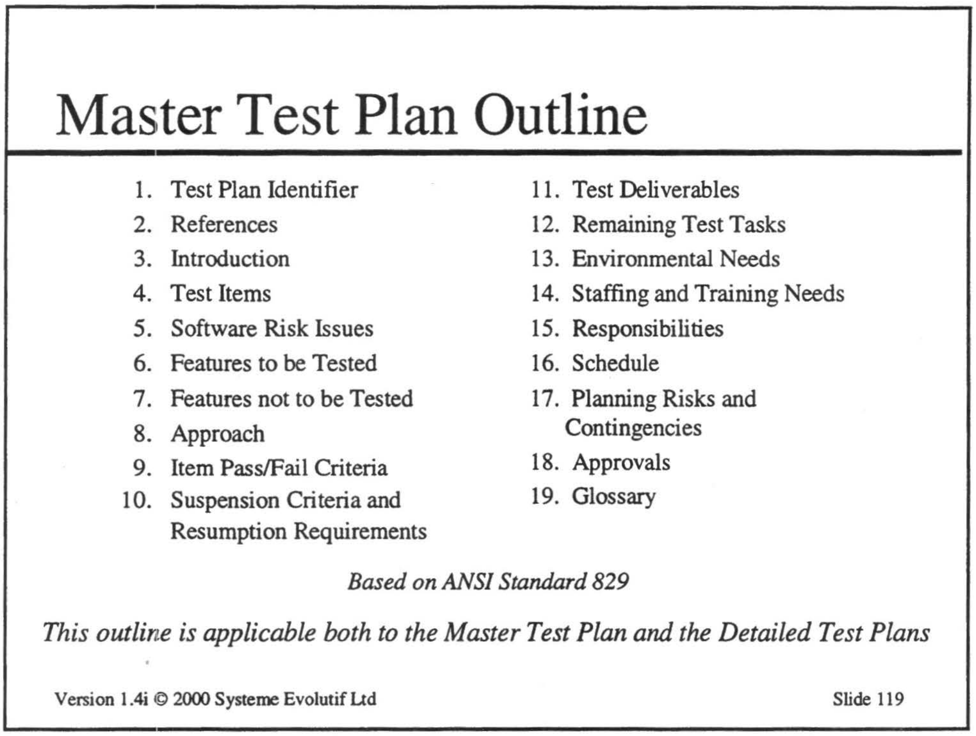


Master Test Plan

- ◆ Addresses project/product and/or individual application/system issues.
- ◆ Focus of strategies, roles, responsibilities, resources, and schedules.
- ◆ The roadmap for all testing activities.
- ◆ Identifies the detailed test plans required.
- ◆ Adopts/adapts test strategy/policies.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 118



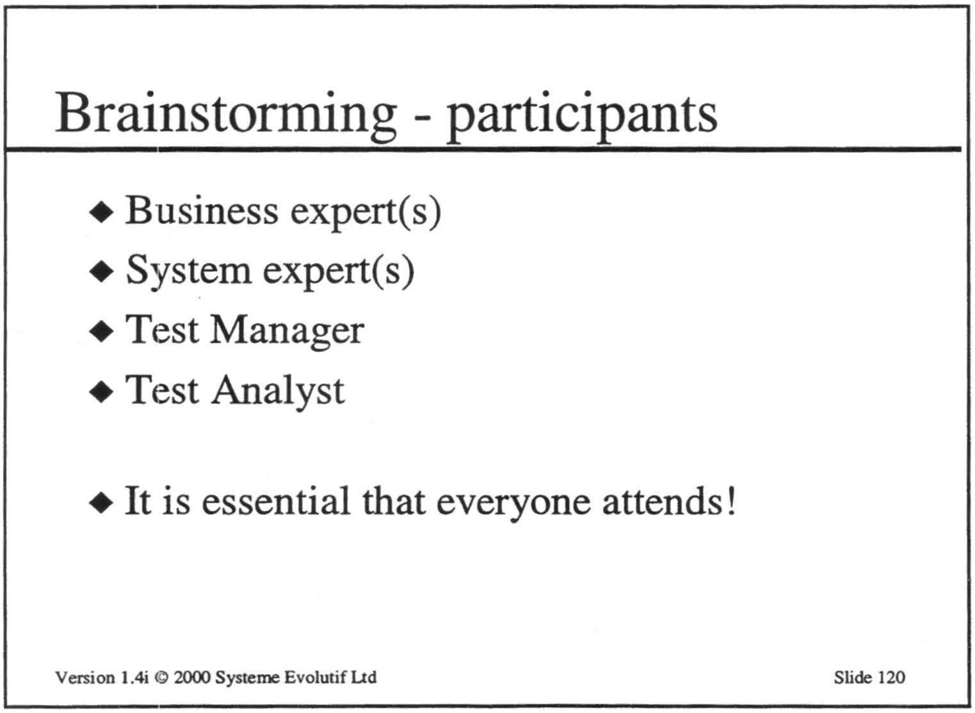
Master Test Plan Outline

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Software Risk Issues
6. Features to be Tested
7. Features not to be Tested
8. Approach
9. Item Pass/Fail Criteria
10. Suspension Criteria and Resumption Requirements
11. Test Deliverables
12. Remaining Test Tasks
13. Environmental Needs
14. Staffing and Training Needs
15. Responsibilities
16. Schedule
17. Planning Risks and Contingencies
18. Approvals
19. Glossary

Based on ANSI Standard 829

This outline is applicable both to the Master Test Plan and the Detailed Test Plans

Version 1.4i © 2000 Systeme Evolutif Ltd   Slide 119

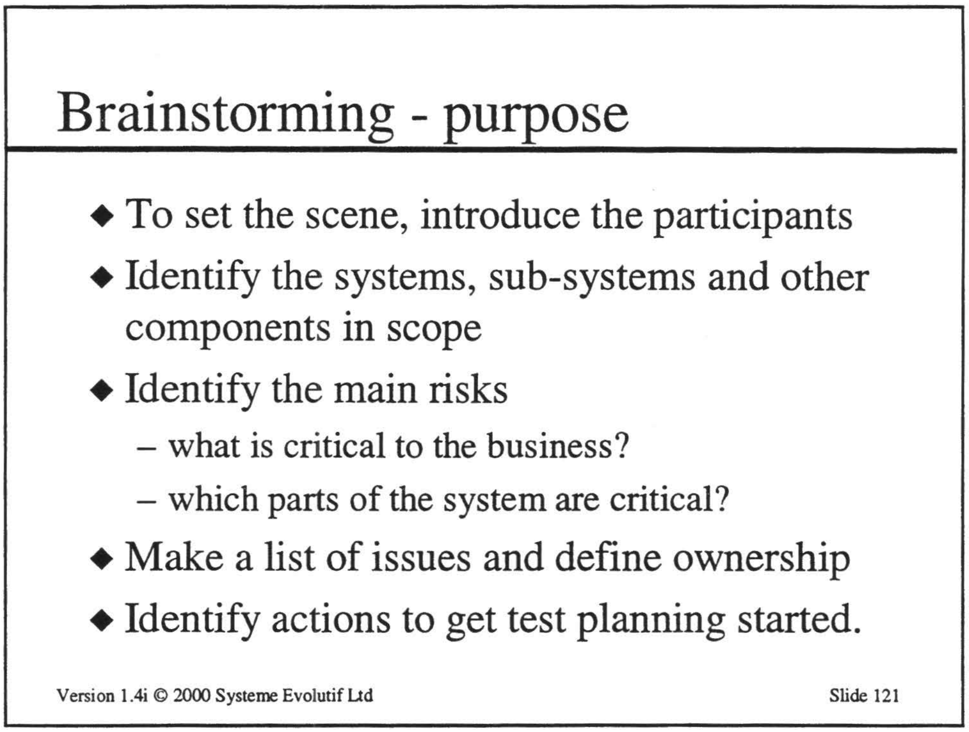


Brainstorming - participants

- ◆ Business expert(s)
- ◆ System expert(s)
- ◆ Test Manager
- ◆ Test Analyst

- ◆ It is essential that everyone attends!

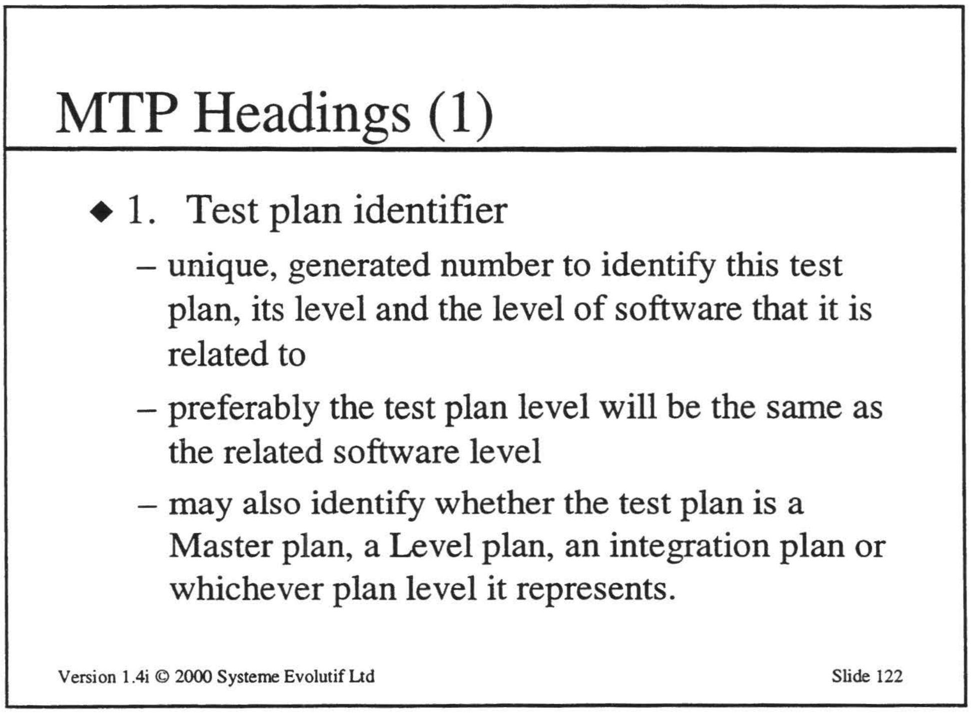
Version 1.4i © 2000 Systeme Evolutif Ltd   Slide 120



Brainstorming - purpose

- To set the scene, introduce the participants
- Identify the systems, sub-systems and other components in scope
- Identify the main risks
  - what is critical to the business?
  - which parts of the system are critical?
- Make a list of issues and define ownership
- Identify actions to get test planning started.

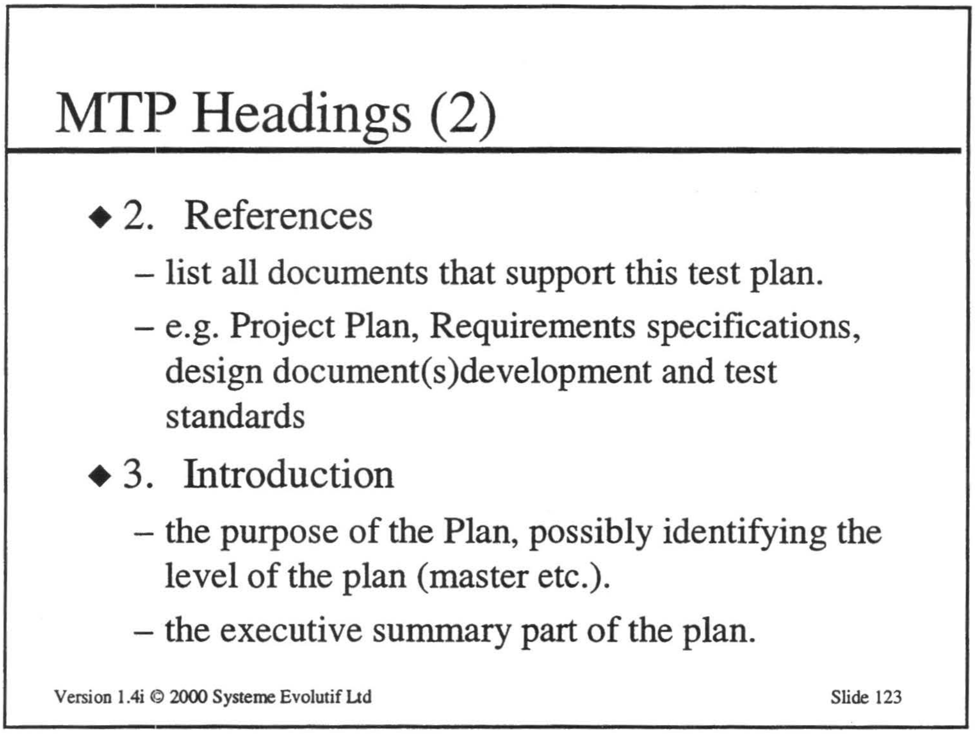
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 121



MTP Headings (1)

- 1. Test plan identifier
  - unique, generated number to identify this test plan, its level and the level of software that it is related to
  - preferably the test plan level will be the same as the related software level
  - may also identify whether the test plan is a Master plan, a Level plan, an integration plan or whichever plan level it represents.

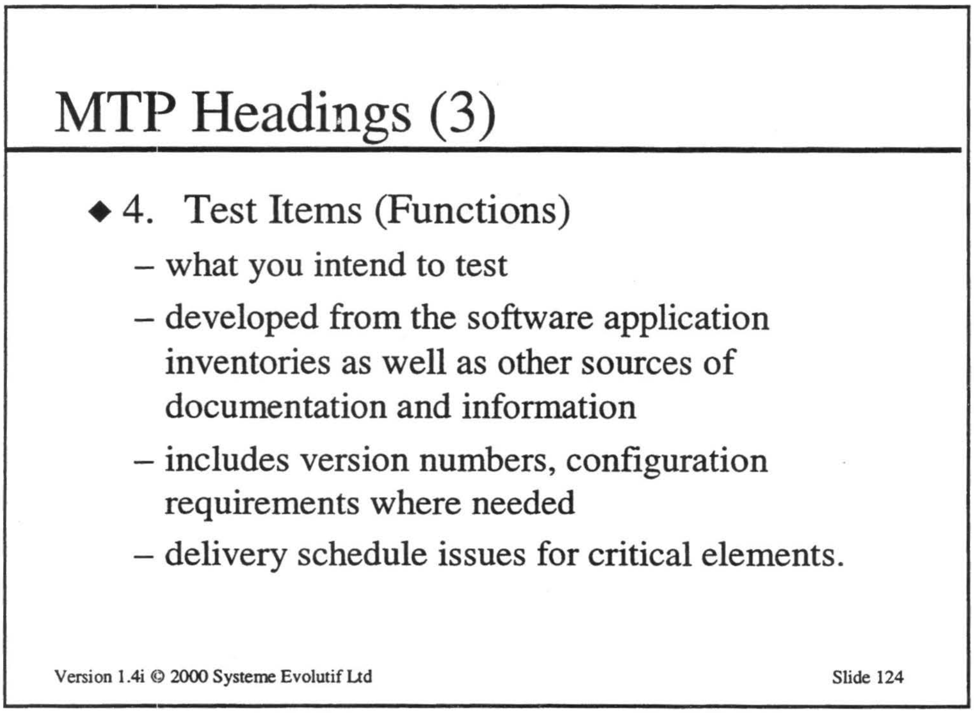
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 122



MTP Headings (2)

- 2. References
  - – list all documents that support this test plan.
  - – e.g. Project Plan, Requirements specifications, design document(s)development and test standards
- 3. Introduction
  - – the purpose of the Plan, possibly identifying the level of the plan (master etc.).
  - – the executive summary part of the plan.

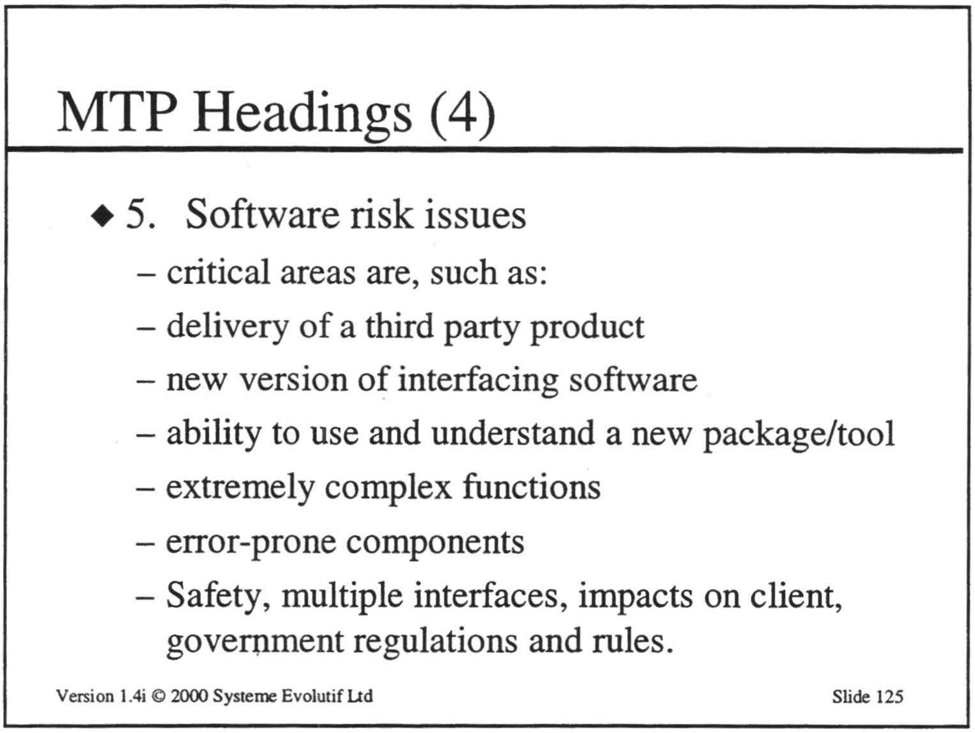
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 123



MTP Headings (3)

- 4. Test Items (Functions)
  - – what you intend to test
  - – developed from the software application inventories as well as other sources of documentation and information
  - – includes version numbers, configuration requirements where needed
  - – delivery schedule issues for critical elements.

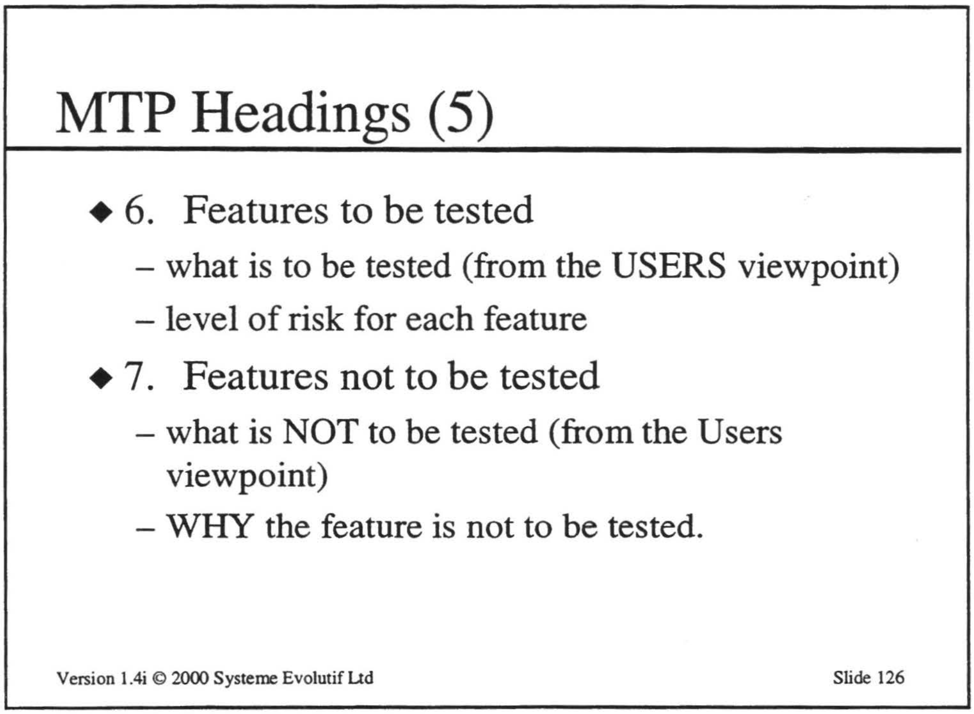
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 124



MTP Headings (4)

- 5. Software risk issues
  - – critical areas are, such as:
  - – delivery of a third party product
  - – new version of interfacing software
  - – ability to use and understand a new package/tool
  - – extremely complex functions
  - – error-prone components
  - – Safety, multiple interfaces, impacts on client, government regulations and rules.

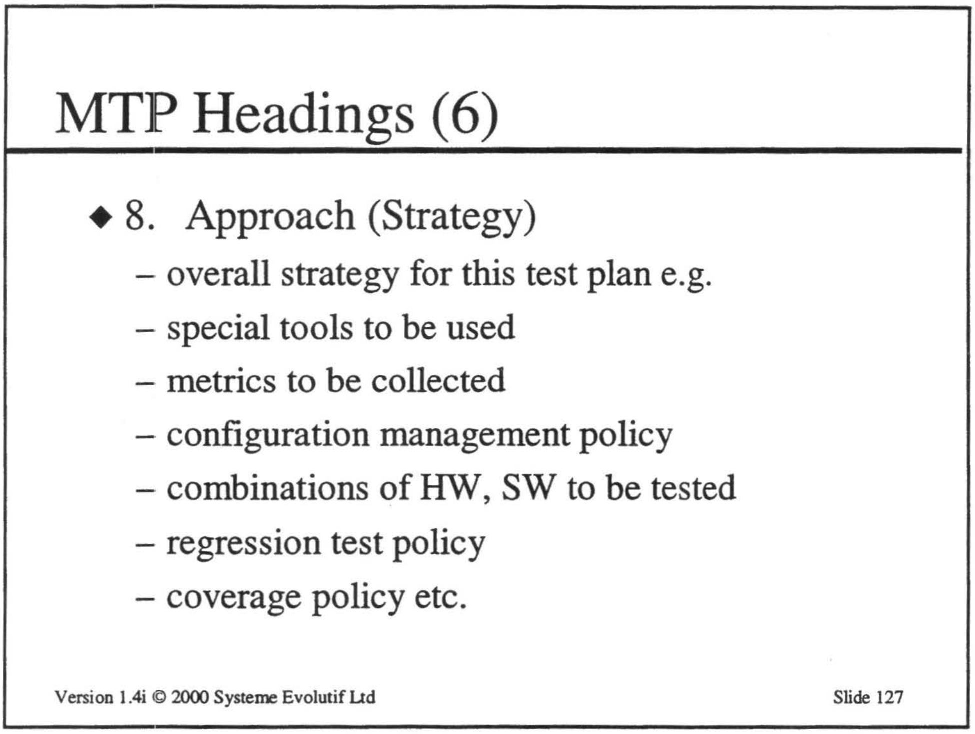
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 125



MTP Headings (5)

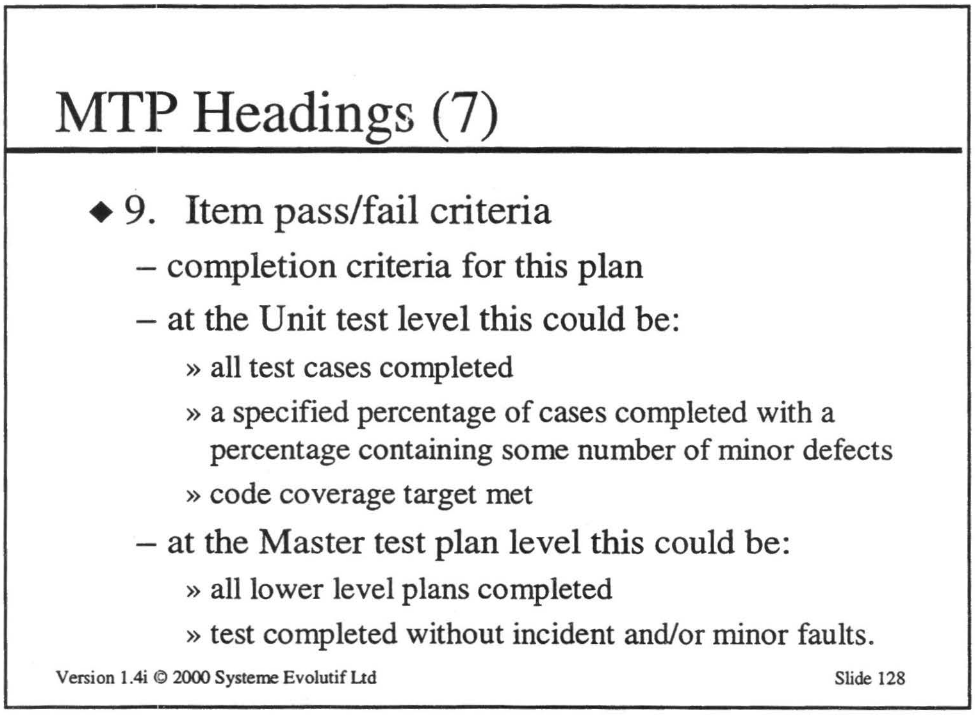
- 6. Features to be tested
  - – what is to be tested (from the USERS viewpoint)
  - – level of risk for each feature
- 7. Features not to be tested
  - – what is NOT to be tested (from the Users viewpoint)
  - – WHY the feature is not to be tested.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 126



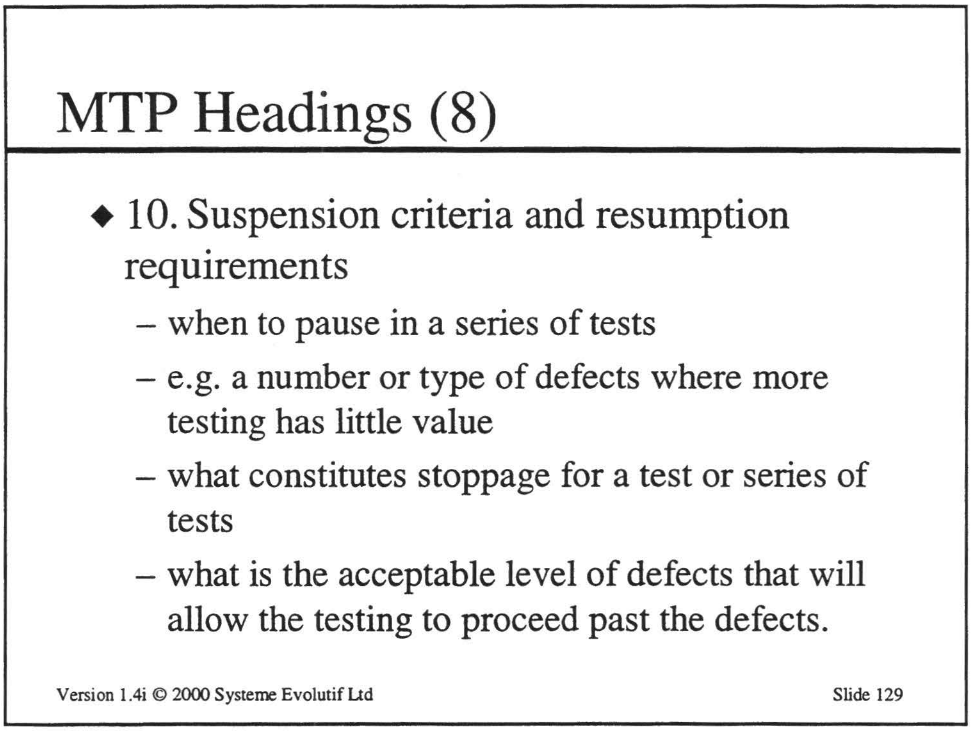
MTP Headings (6)

- 8. Approach (Strategy)
  - overall strategy for this test plan e.g.
  - special tools to be used
  - metrics to be collected
  - configuration management policy
  - combinations of HW, SW to be tested
  - regression test policy
  - coverage policy etc.



MTP Headings (7)

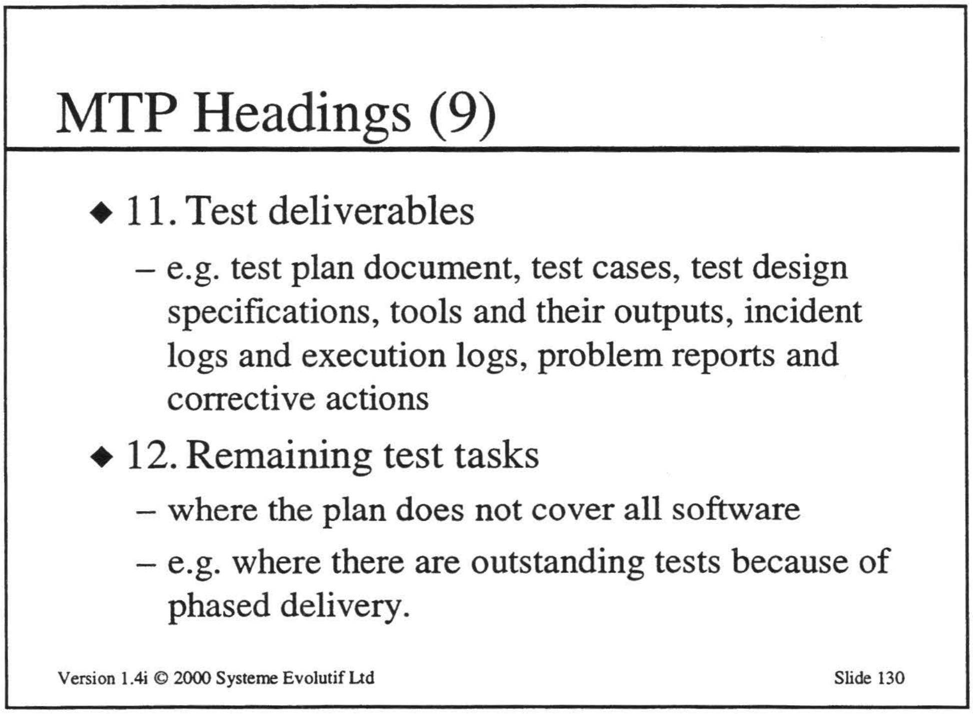
- 9. Item pass/fail criteria
  - completion criteria for this plan
  - at the Unit test level this could be:
    - » all test cases completed
    - » a specified percentage of cases completed with a percentage containing some number of minor defects
    - » code coverage target met
  - at the Master test plan level this could be:
    - » all lower level plans completed
    - » test completed without incident and/or minor faults.



MTP Headings (8)

- 10. Suspension criteria and resumption requirements
  - when to pause in a series of tests
  - e.g. a number or type of defects where more testing has little value
  - what constitutes stoppage for a test or series of tests
  - what is the acceptable level of defects that will allow the testing to proceed past the defects.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 129



MTP Headings (9)

- 11. Test deliverables
  - e.g. test plan document, test cases, test design specifications, tools and their outputs, incident logs and execution logs, problem reports and corrective actions
- 12. Remaining test tasks
  - where the plan does not cover all software
  - e.g. where there are outstanding tests because of phased delivery.

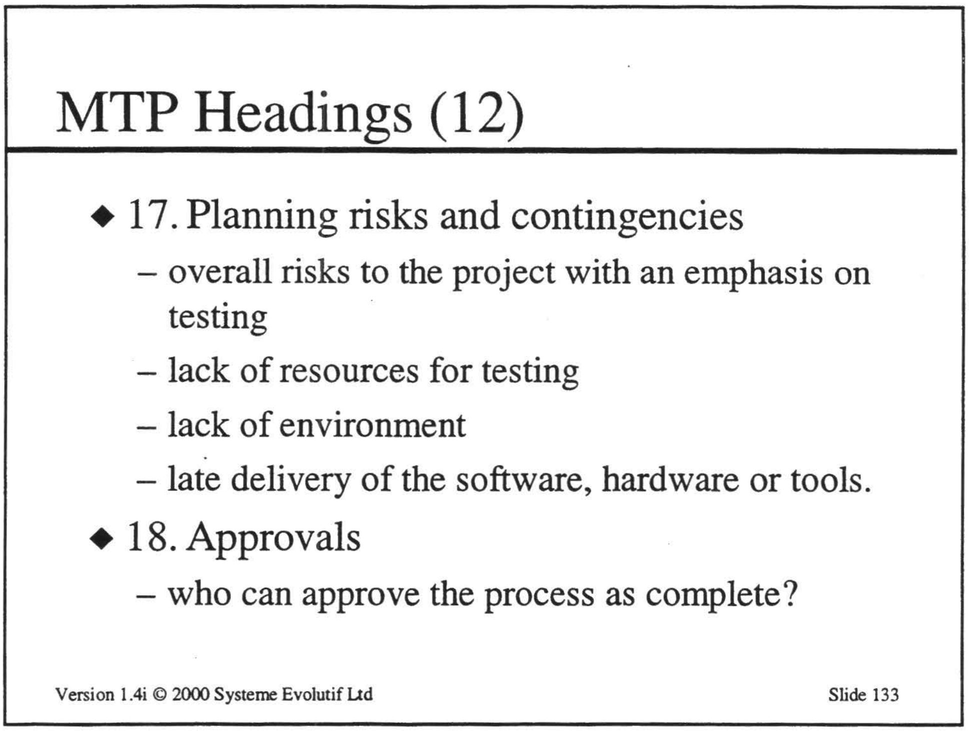
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 130

MTP Headings (10)

- 13. Environmental needs
  - special requirements such as:
    - » special hardware such as simulators, test drivers etc.
    - » how test data will be provided
- 14. Staffing and training
  - e.g. training on the application/system
  - training for any test tools to be used.

MTP Headings (11)

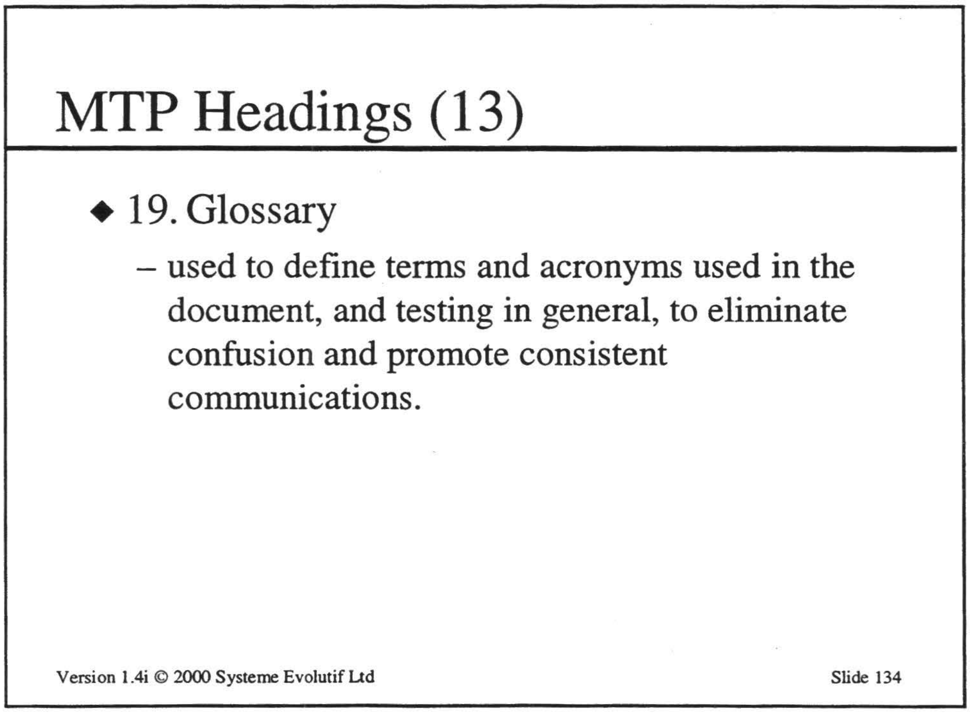
- 15. Responsibilities
  - who is in charge?
  - who defines the risks?
  - who selects features to be tested and not tested
  - who sets overall strategy for this level of plan.
- 16. Schedule
  - based on realistic and validated estimates.



MTP Headings (12)

- 17. Planning risks and contingencies
  - overall risks to the project with an emphasis on testing
  - lack of resources for testing
  - lack of environment
  - late delivery of the software, hardware or tools.
- 18. Approvals
  - who can approve the process as complete?

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 133



MTP Headings (13)

- 19. Glossary
  - used to define terms and acronyms used in the document, and testing in general, to eliminate confusion and promote consistent communications.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 134

Testability

Version 1.4i © 2000 Systeme Evolutif Ltd

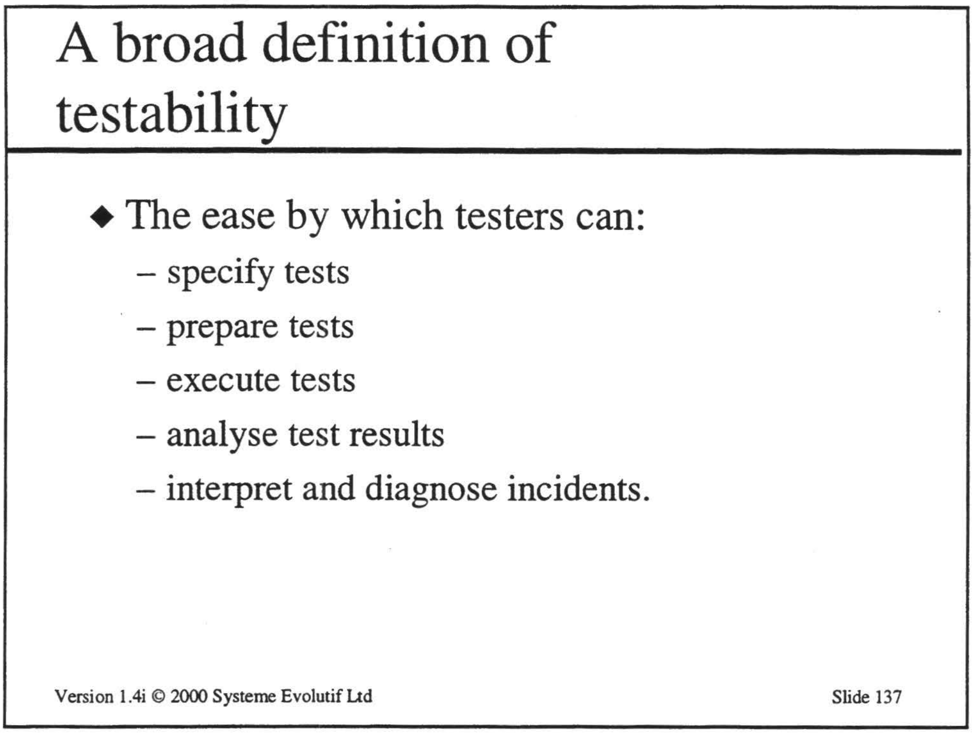
Slide 135

Testability definitions (testable requirements)

- “The extent to which software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria” or
- “The extent to which the definition of requirements facilitates analysis of the requirements to establish test criteria.”

Version 1.4i © 2000 Systeme Evolutif Ltd

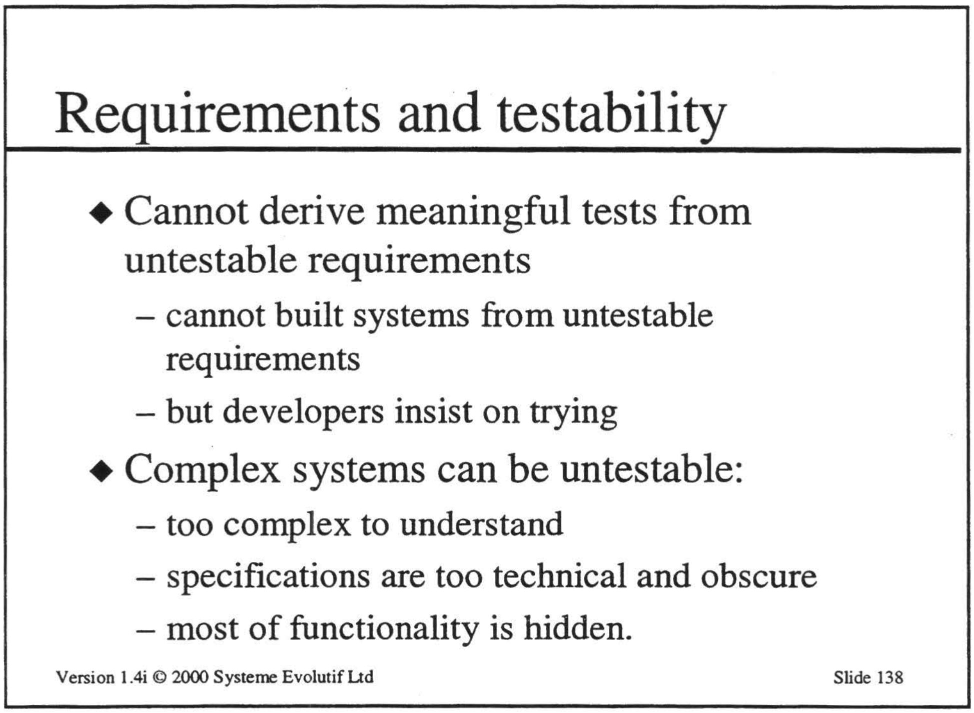
Slide 136



A broad definition of testability

- The ease by which testers can:
  - specify tests
  - prepare tests
  - execute tests
  - analyse test results
  - interpret and diagnose incidents.

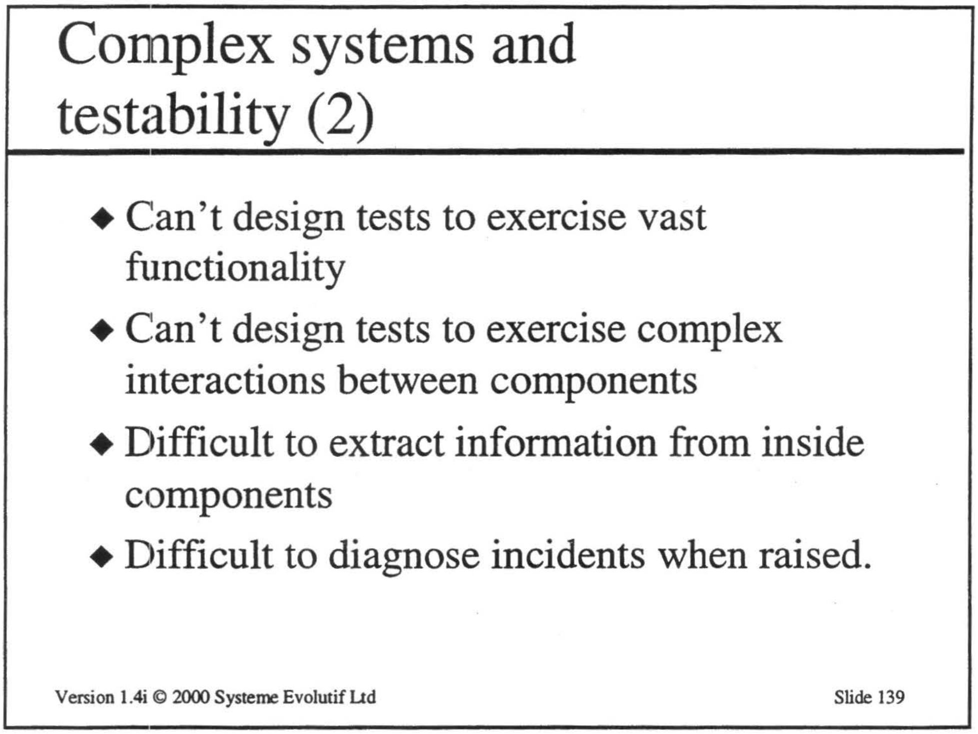
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 137



Requirements and testability

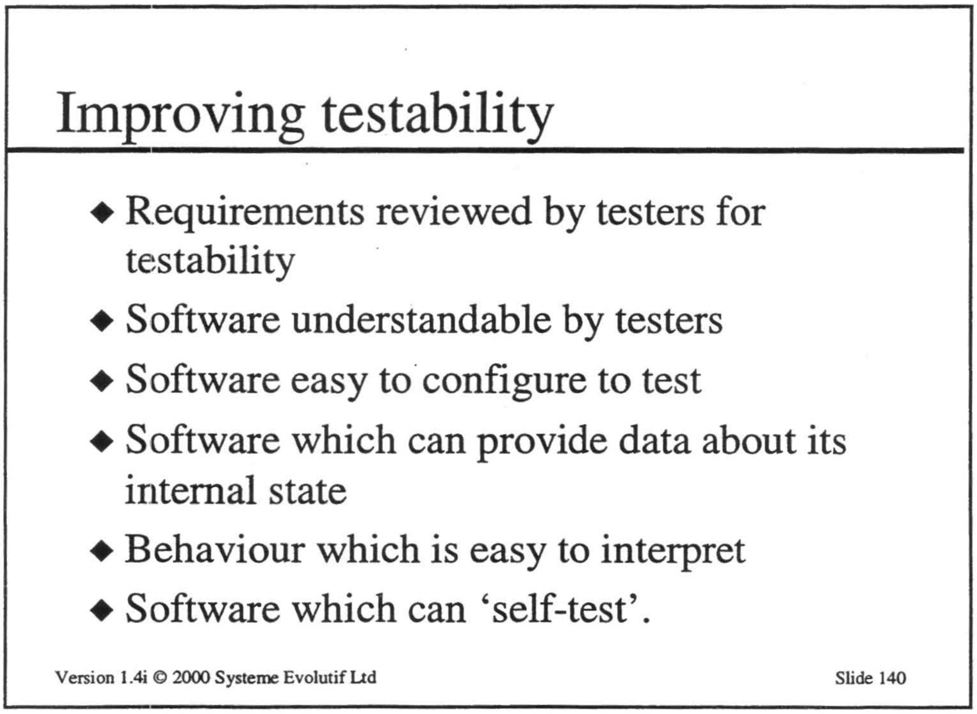
- Cannot derive meaningful tests from untestable requirements
  - cannot built systems from untestable requirements
  - but developers insist on trying
- Complex systems can be untestable:
  - too complex to understand
  - specifications are too technical and obscure
  - most of functionality is hidden.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 138



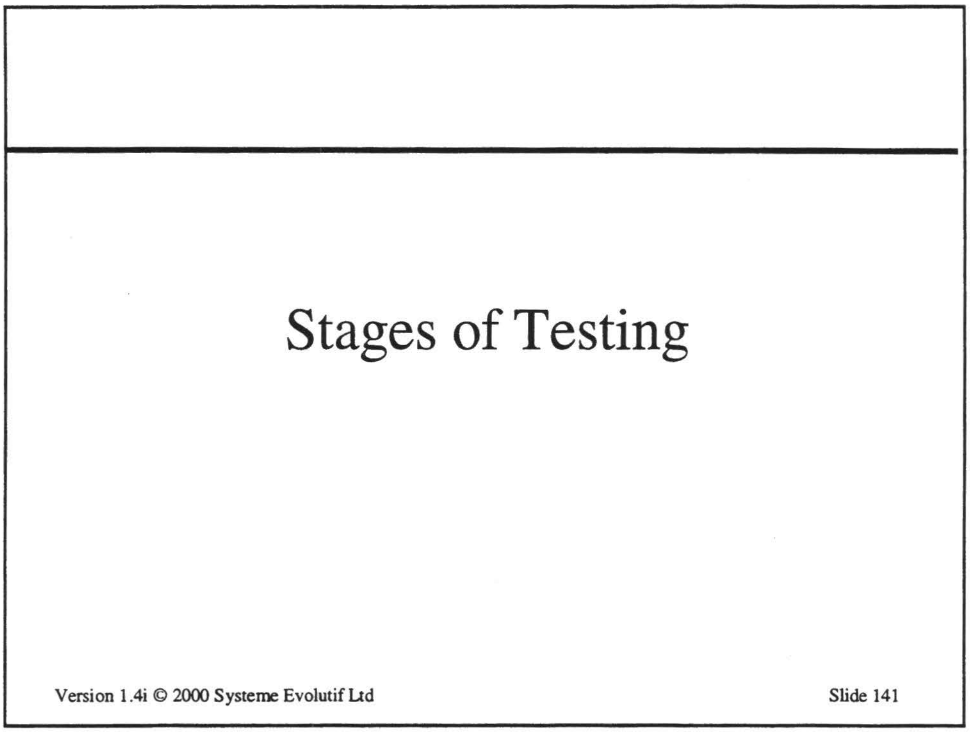
Complex systems and testability (2)

- Can’t design tests to exercise vast functionality
- Can’t design tests to exercise complex interactions between components
- Difficult to extract information from inside components
- Difficult to diagnose incidents when raised.



Improving testability

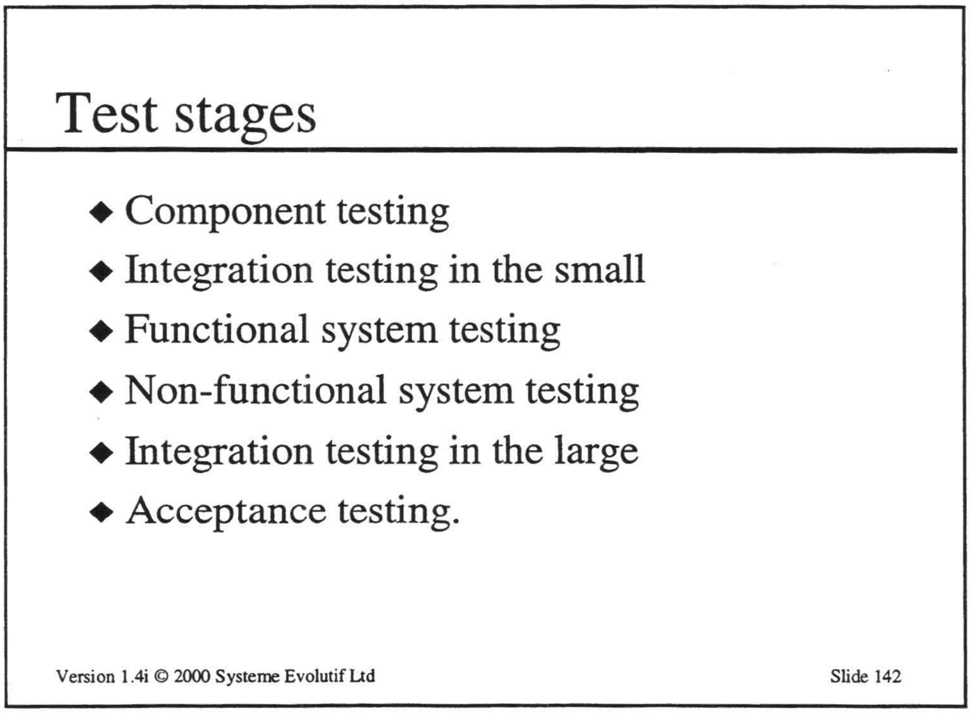
- Requirements reviewed by testers for testability
- Software understandable by testers
- Software easy to configure to test
- Software which can provide data about its internal state
- Behaviour which is easy to interpret
- Software which can ‘self-test’.



Stages of Testing

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 141

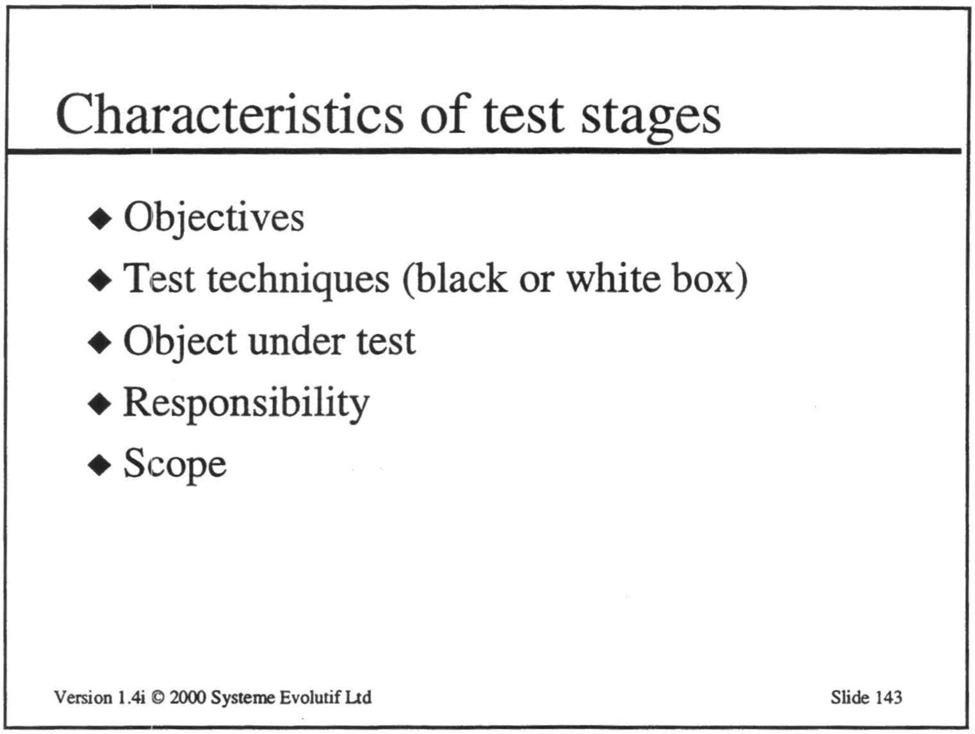


Test stages

- ◆ Component testing
- ◆ Integration testing in the small
- ◆ Functional system testing
- ◆ Non-functional system testing
- ◆ Integration testing in the large
- ◆ Acceptance testing.

Version 1.4i © 2000 Systeme Evolutif Ltd

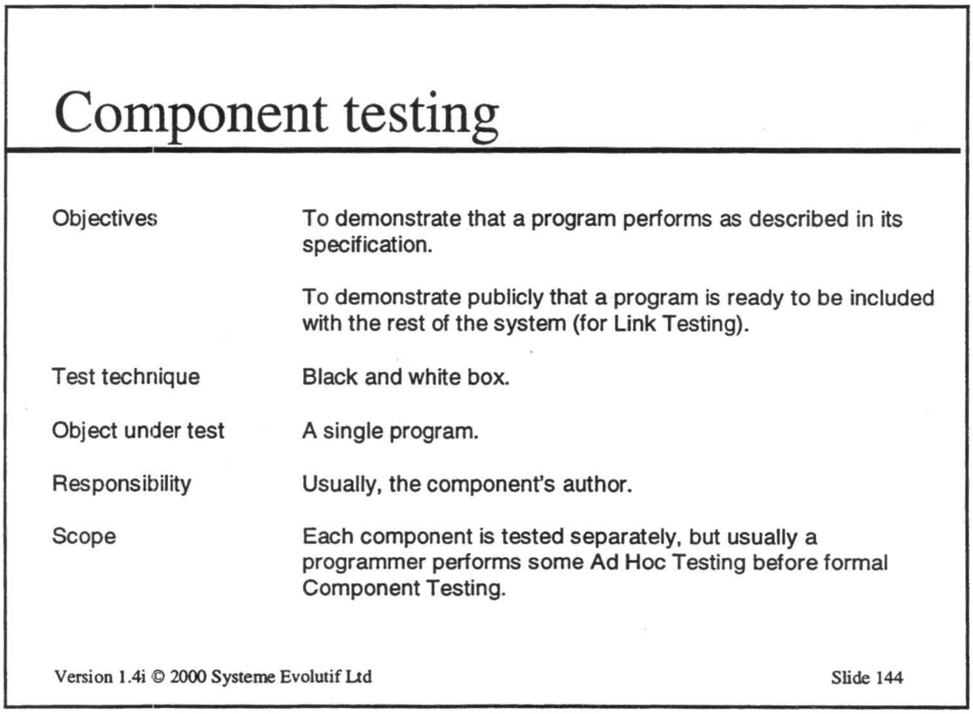
Slide 142



Characteristics of test stages

- ◆ Objectives
- ◆ Test techniques (black or white box)
- ◆ Object under test
- ◆ Responsibility
- ◆ Scope

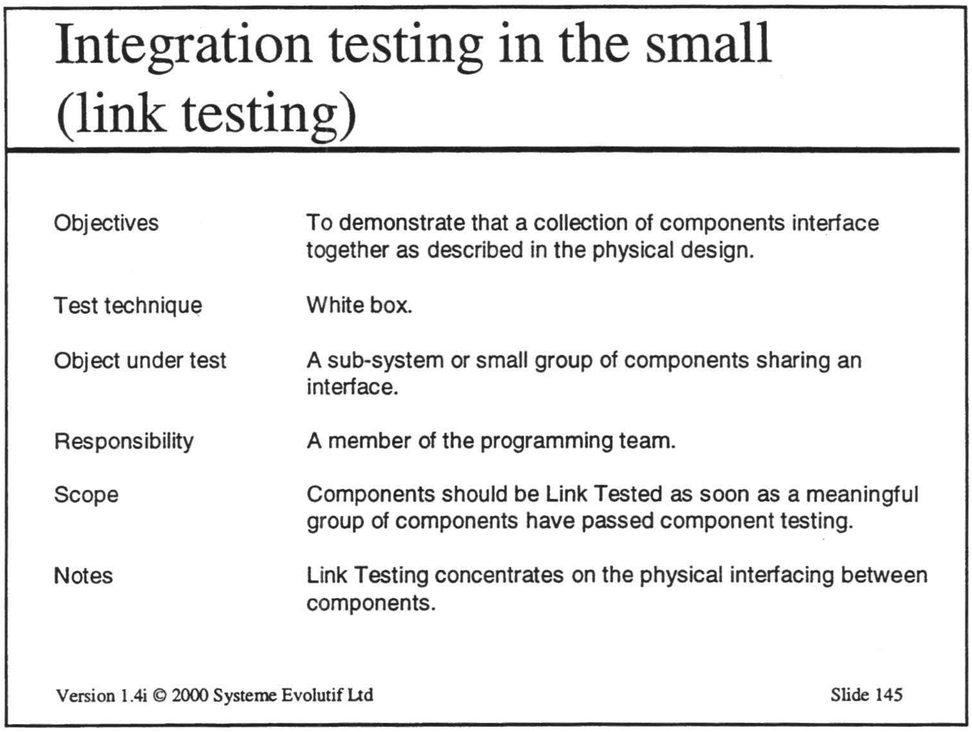
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 143



Component testing

Objectives	To demonstrate that a program performs as described in its specification. To demonstrate publicly that a program is ready to be included with the rest of the system (for Link Testing).
Test technique	Black and white box.
Object under test	A single program.
Responsibility	Usually, the component's author.
Scope	Each component is tested separately, but usually a programmer performs some Ad Hoc Testing before formal Component Testing.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 144

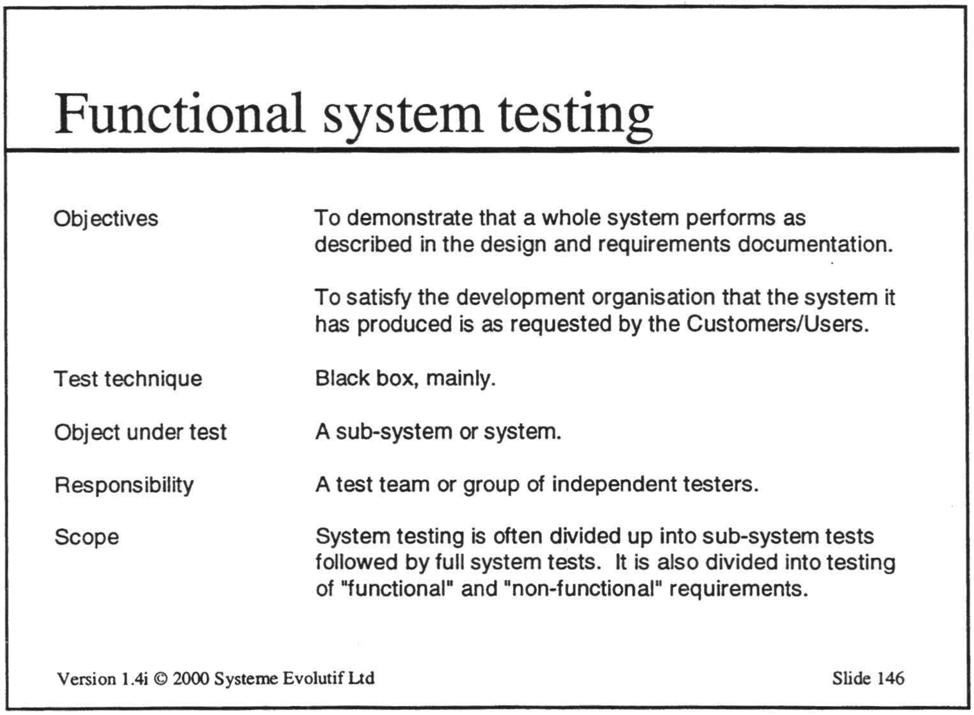


Integration testing in the small (link testing)

Objectives	To demonstrate that a collection of components interface together as described in the physical design.
Test technique	White box.
Object under test	A sub-system or small group of components sharing an interface.
Responsibility	A member of the programming team.
Scope	Components should be Link Tested as soon as a meaningful group of components have passed component testing.
Notes	Link Testing concentrates on the physical interfacing between components.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 145

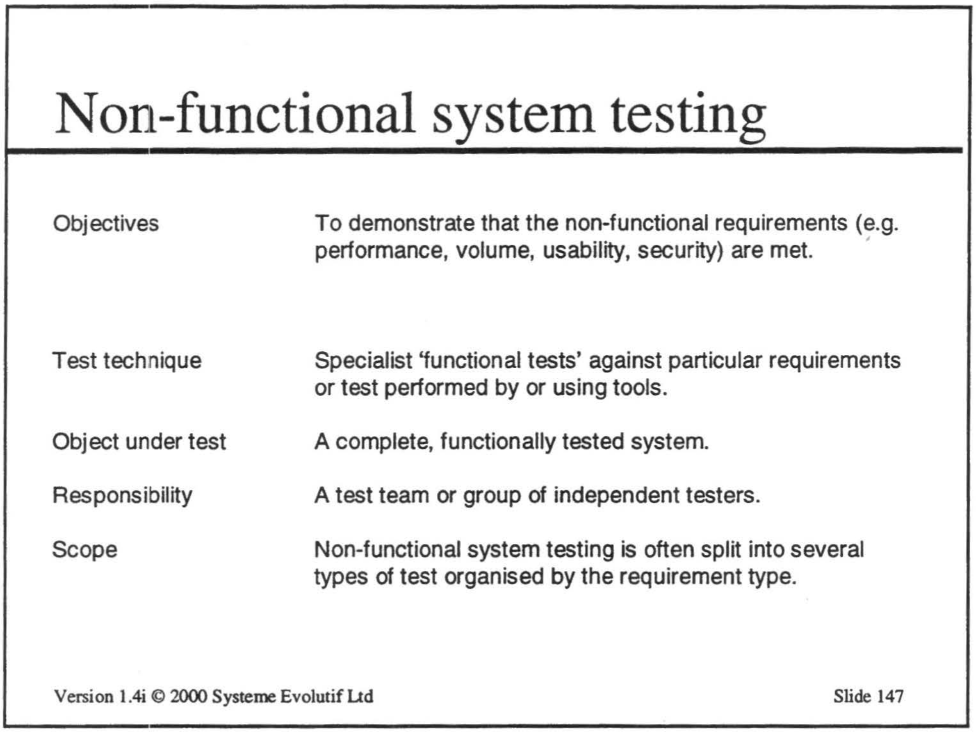


Functional system testing

Objectives	To demonstrate that a whole system performs as described in the design and requirements documentation. To satisfy the development organisation that the system it has produced is as requested by the Customers/Users.
Test technique	Black box, mainly.
Object under test	A sub-system or system.
Responsibility	A test team or group of independent testers.
Scope	System testing is often divided up into sub-system tests followed by full system tests. It is also divided into testing of "functional" and "non-functional" requirements.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 146



Non-functional system testing

Objectives — To demonstrate that the non-functional requirements (e.g. performance, volume, usability, security) are met.

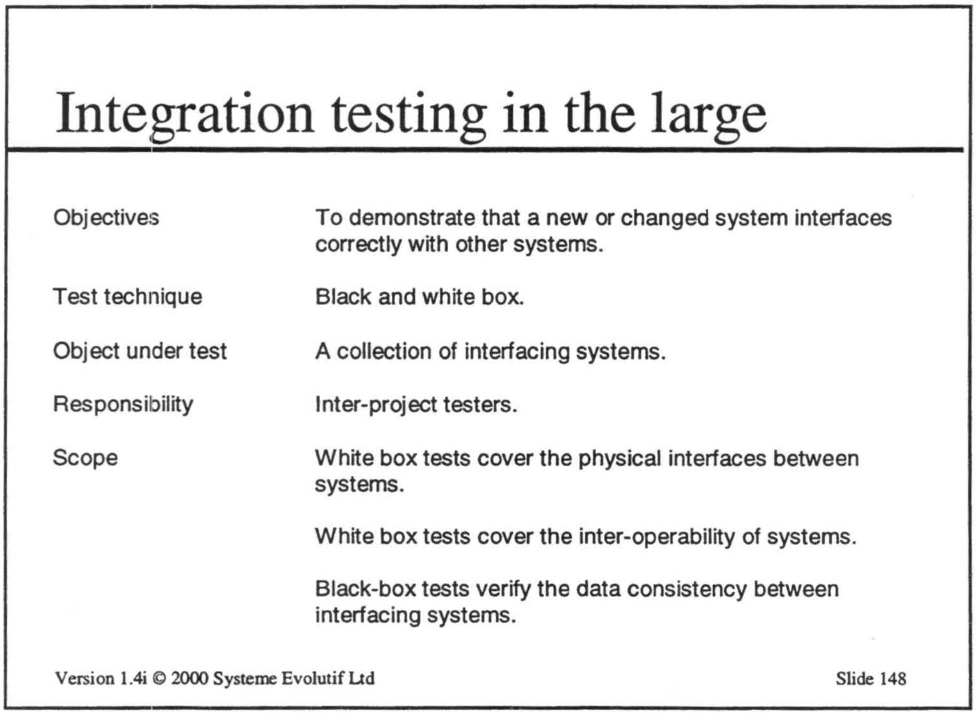
Test technique — Specialist 'functional tests' against particular requirements or test performed by or using tools.

Object under test — A complete, functionally tested system.

Responsibility — A test team or group of independent testers.

Scope — Non-functional system testing is often split into several types of test organised by the requirement type.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 147



Integration testing in the large

Objectives — To demonstrate that a new or changed system interfaces correctly with other systems.

Test technique — Black and white box.

Object under test — A collection of interfacing systems.

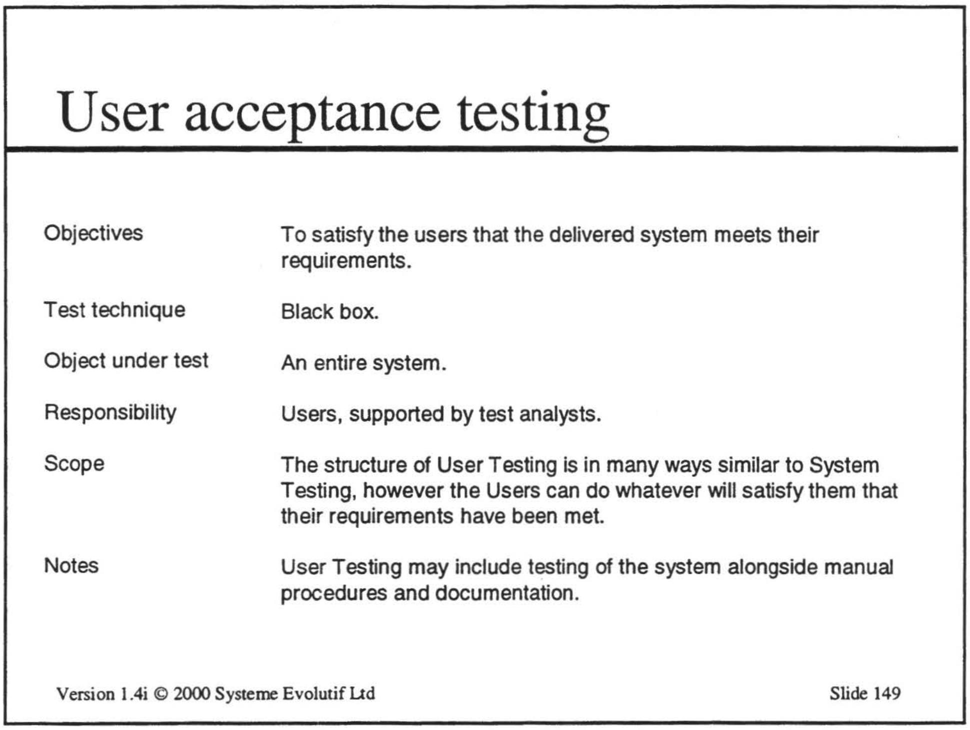
Responsibility — Inter-project testers.

Scope — White box tests cover the physical interfaces between systems.

White box tests cover the inter-operability of systems.

Black-box tests verify the data consistency between interfacing systems.

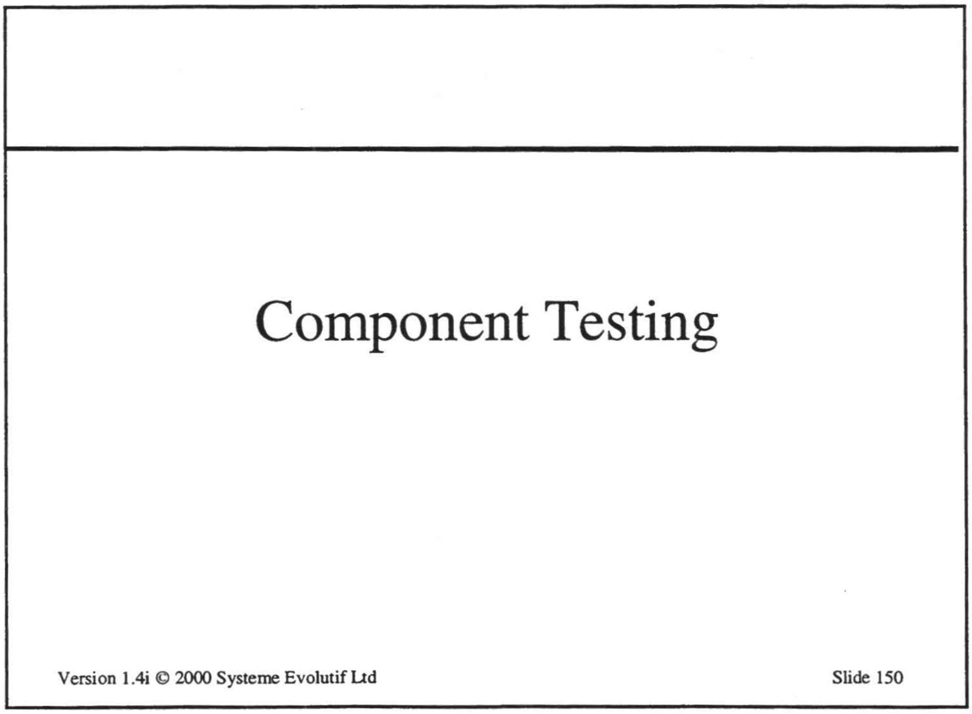
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 148



User acceptance testing

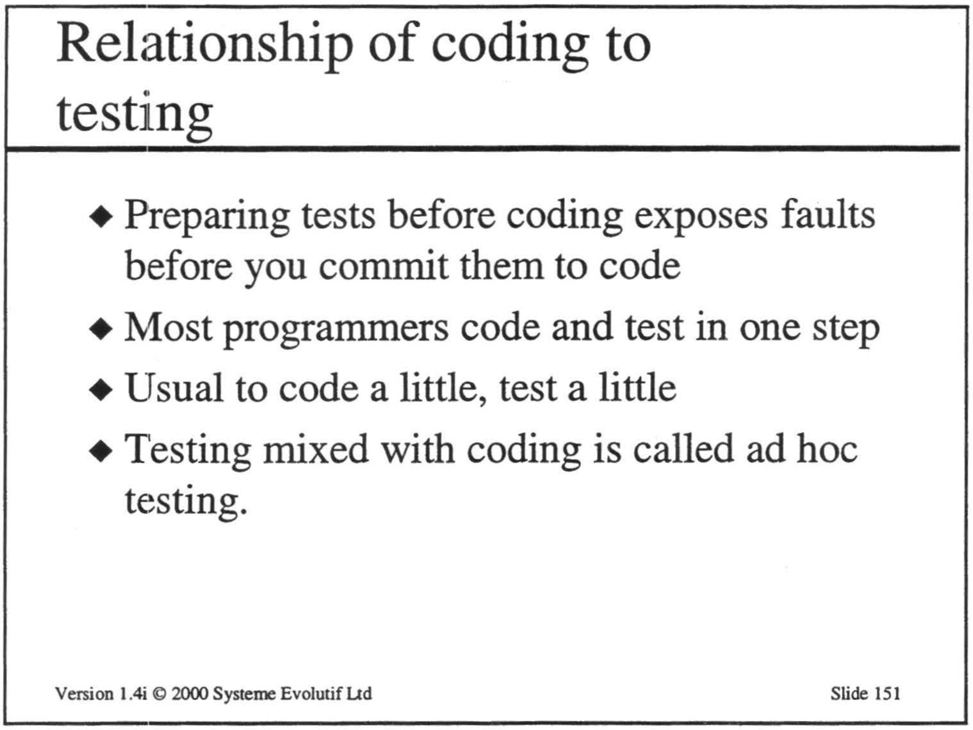
Objectives	To satisfy the users that the delivered system meets their requirements.
Test technique	Black box.
Object under test	An entire system.
Responsibility	Users, supported by test analysts.
Scope	The structure of User Testing is in many ways similar to System Testing, however the Users can do whatever will satisfy them that their requirements have been met.
Notes	User Testing may include testing of the system alongside manual procedures and documentation.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 149



Component Testing

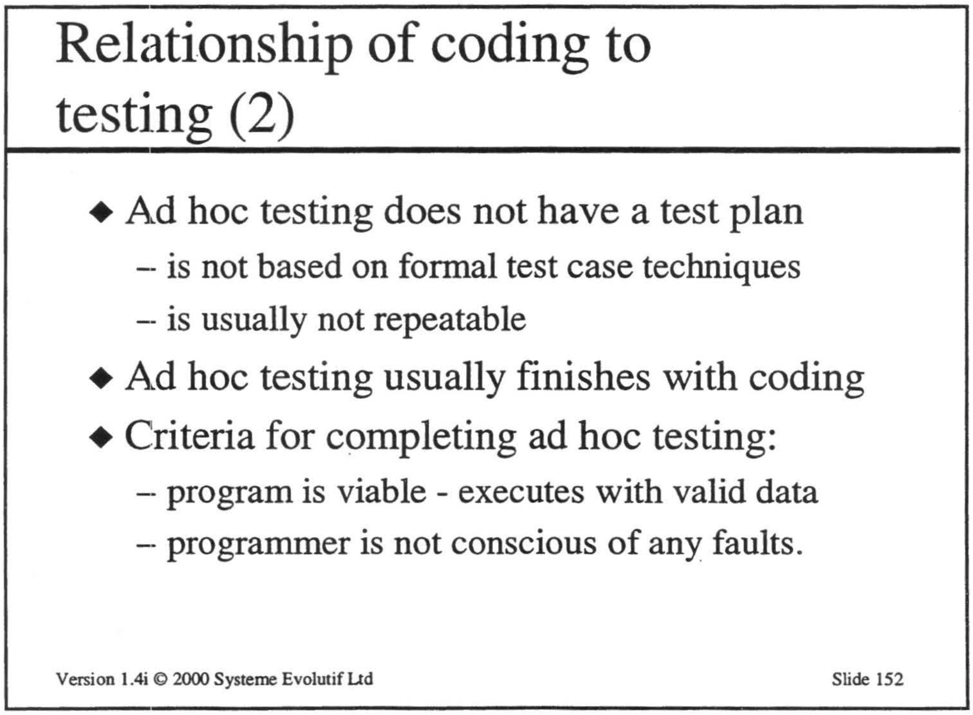
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 150



Relationship of coding to testing

- ◆ Preparing tests before coding exposes faults before you commit them to code
- ◆ Most programmers code and test in one step
- ◆ Usual to code a little, test a little
- ◆ Testing mixed with coding is called ad hoc testing.

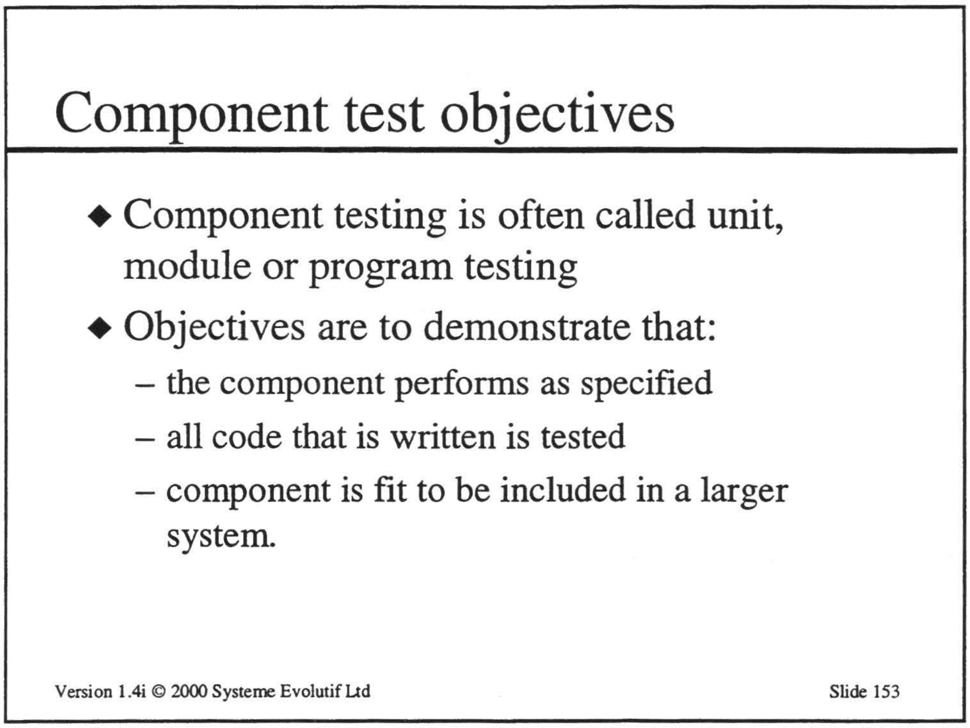
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 151



Relationship of coding to testing (2)

- ◆ Ad hoc testing does not have a test plan
  - – is not based on formal test case techniques
  - – is usually not repeatable
- ◆ Ad hoc testing usually finishes with coding
- ◆ Criteria for completing ad hoc testing:
  - – program is viable - executes with valid data
  - – programmer is not conscious of any faults.

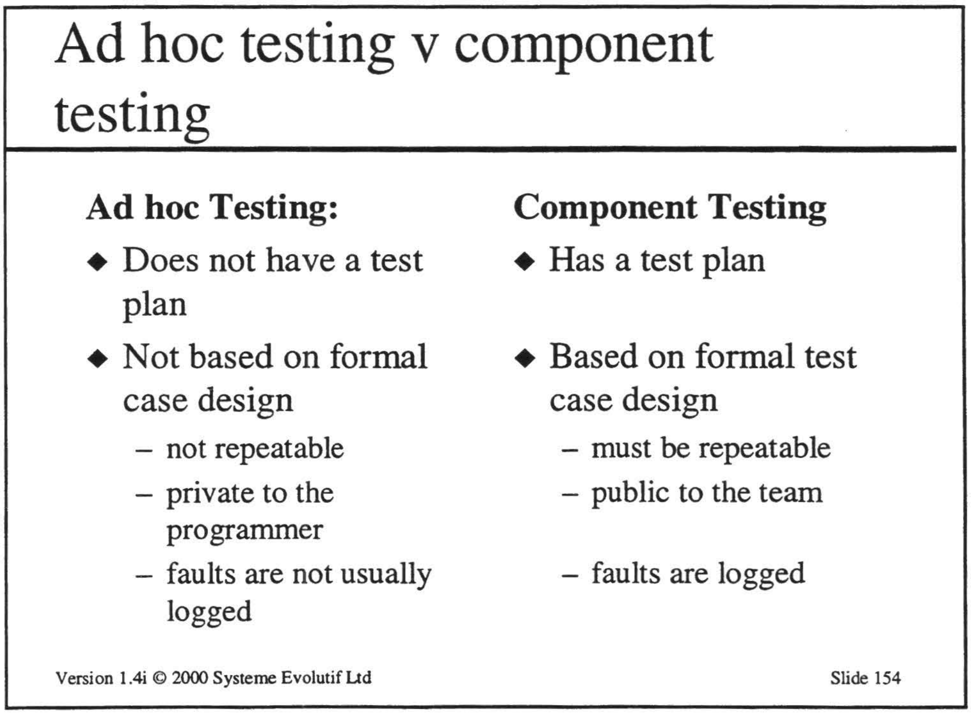
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 152



Component test objectives

- ◆ Component testing is often called unit, module or program testing
- ◆ Objectives are to demonstrate that:
  - – the component performs as specified
  - – all code that is written is tested
  - – component is fit to be included in a larger system.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 153



Ad hoc testing v component testing

Ad hoc Testing:	**Component Testing**
◆ Does not have a test plan	◆ Has a test plan
◆ Not based on formal case design	◆ Based on formal test case design
– not repeatable	– must be repeatable
– private to the programmer	– public to the team
– faults are not usually logged	– faults are logged

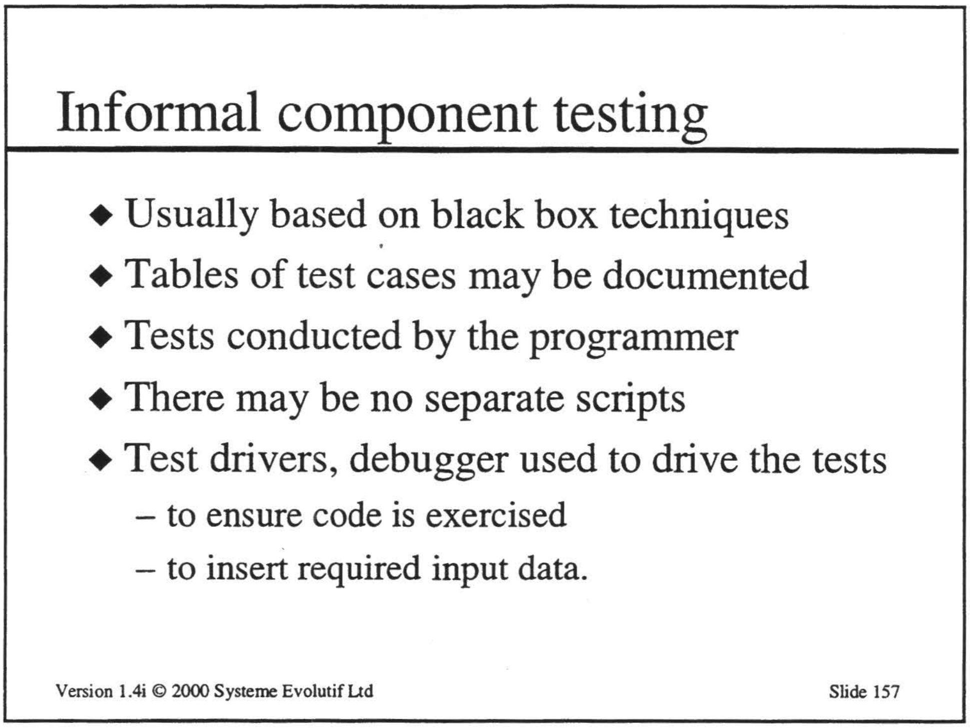
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 154

Analysing a component specification

- Specification reviewers ask ‘how would we test this requirement’ among other questions
- If specifications aren’t reviewed, the programmer is the first person to ‘test’ the specification
  - look for ambiguities, inconsistencies, omissions
  - omissions are hardest to spot
  - preparing tests from specifications finds faults in specifications.

Analysing a component specification (2)

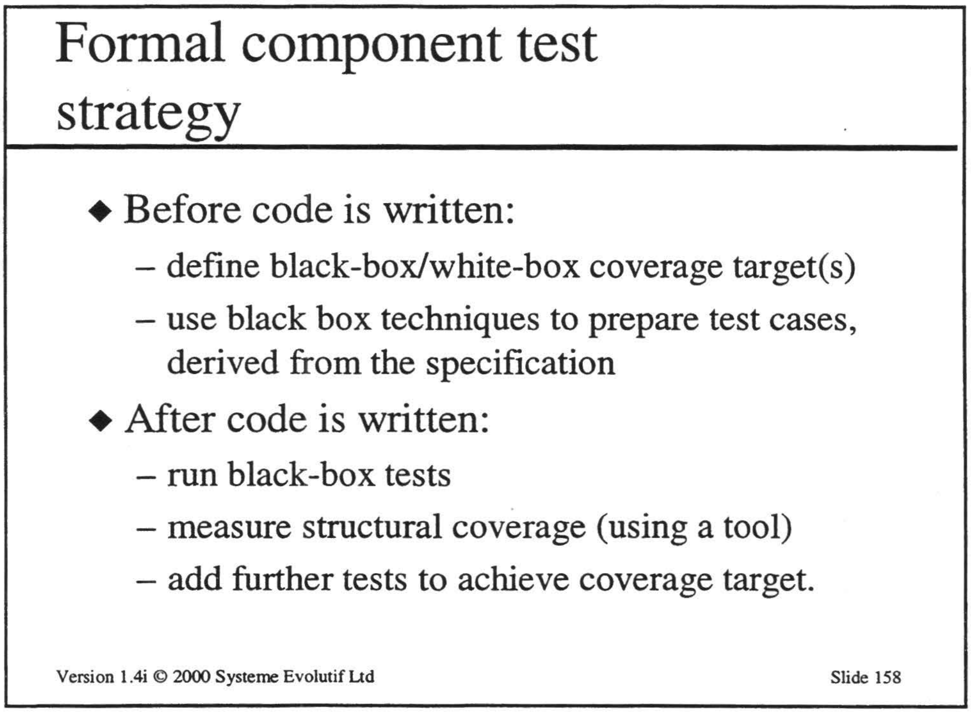
- Get clarification from the author
  - informal walkthroughs
  - explains your understanding of the specification
- May look obvious how to build the program, but is it obvious how to test?
  - if you couldn’t test it, can you *really* build it?
  - how will you demonstrate completion/success?



Informal component testing

- Usually based on black box techniques
- Tables of test cases may be documented
- Tests conducted by the programmer
- There may be no separate scripts
- Test drivers, debugger used to drive the tests
  - to ensure code is exercised
  - to insert required input data.

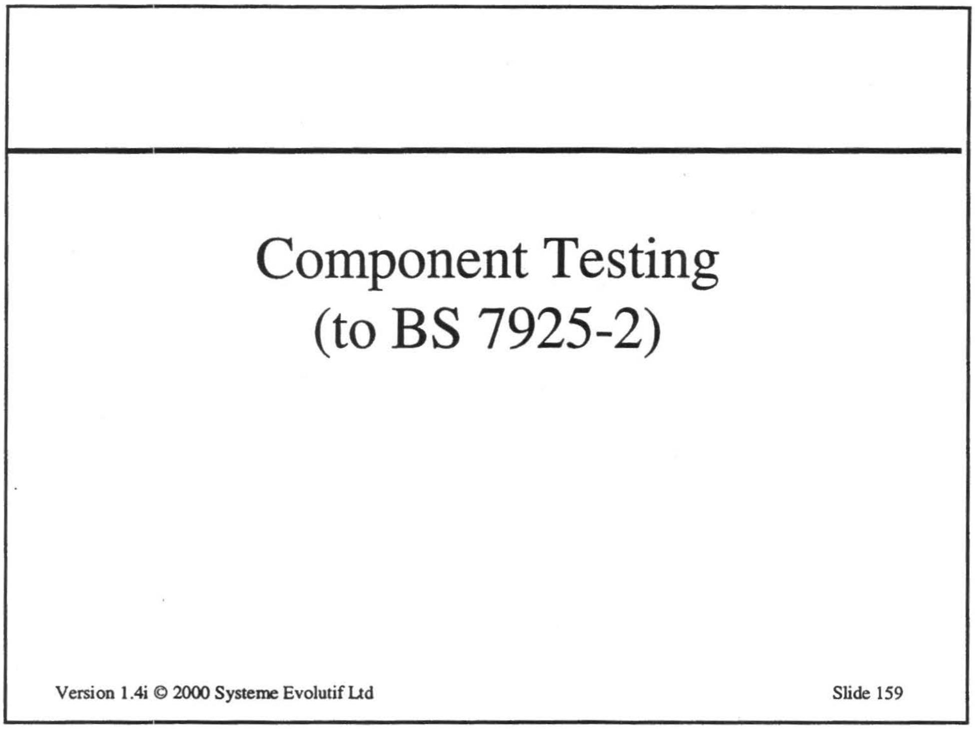
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 157



Formal component test strategy

- Before code is written:
  - define black-box/white-box coverage target(s)
  - use black box techniques to prepare test cases, derived from the specification
- After code is written:
  - run black-box tests
  - measure structural coverage (using a tool)
  - add further tests to achieve coverage target.

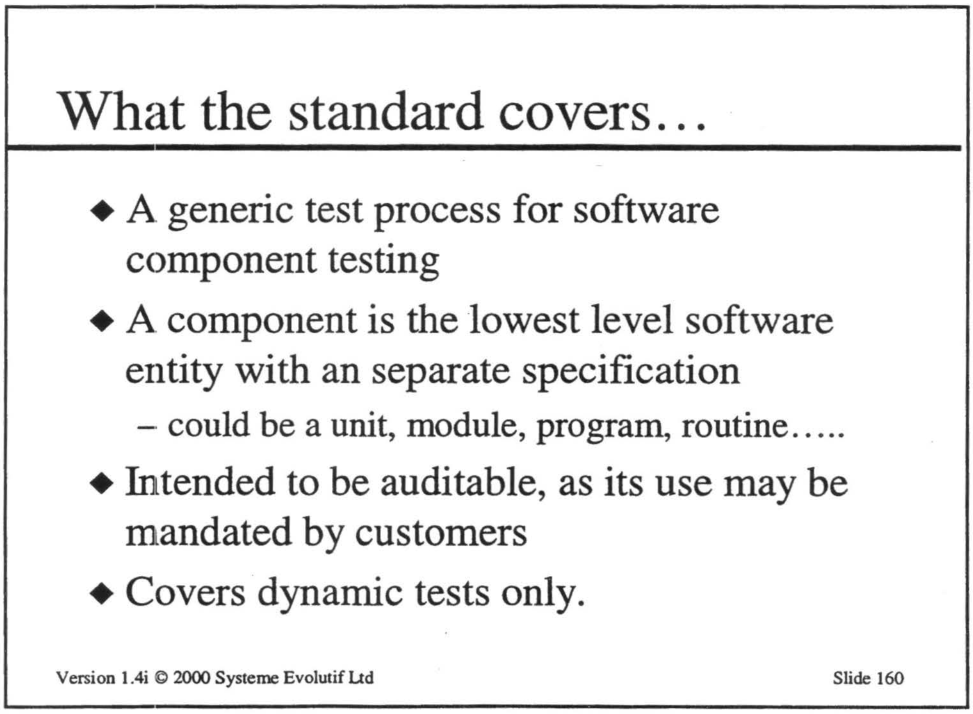
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 158



Component Testing (to BS 7925-2)

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 159

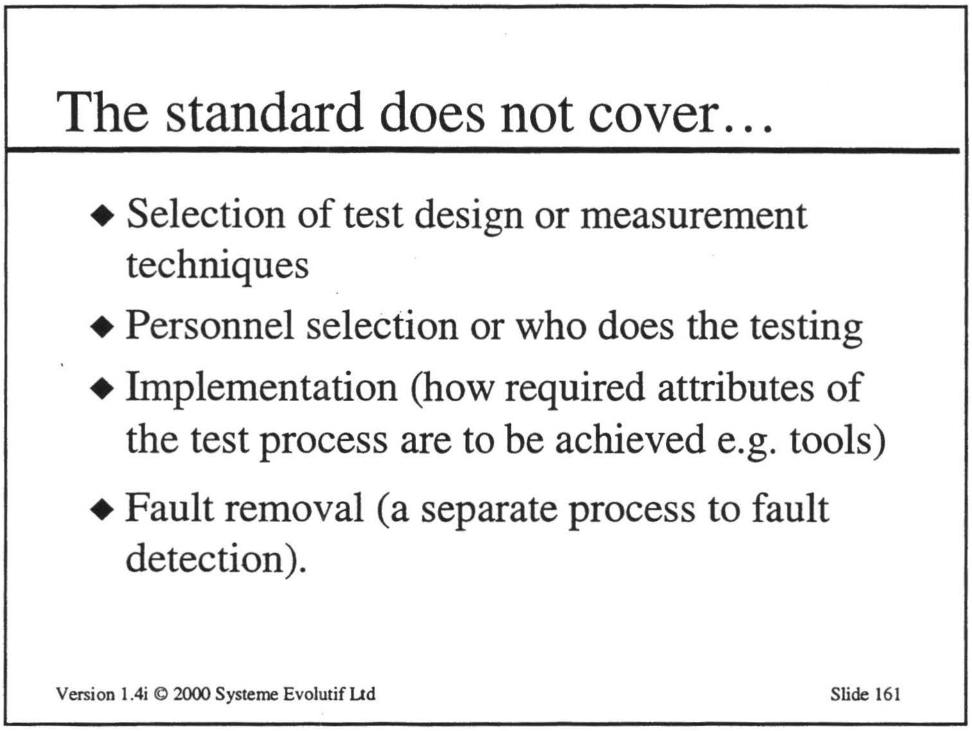


What the standard covers...

- A generic test process for software component testing
- A component is the lowest level software entity with an separate specification
  - could be a unit, module, program, routine.....
- Intended to be auditable, as its use may be mandated by customers
- Covers dynamic tests only.

Version 1.4i © 2000 Systeme Evolutif Ltd

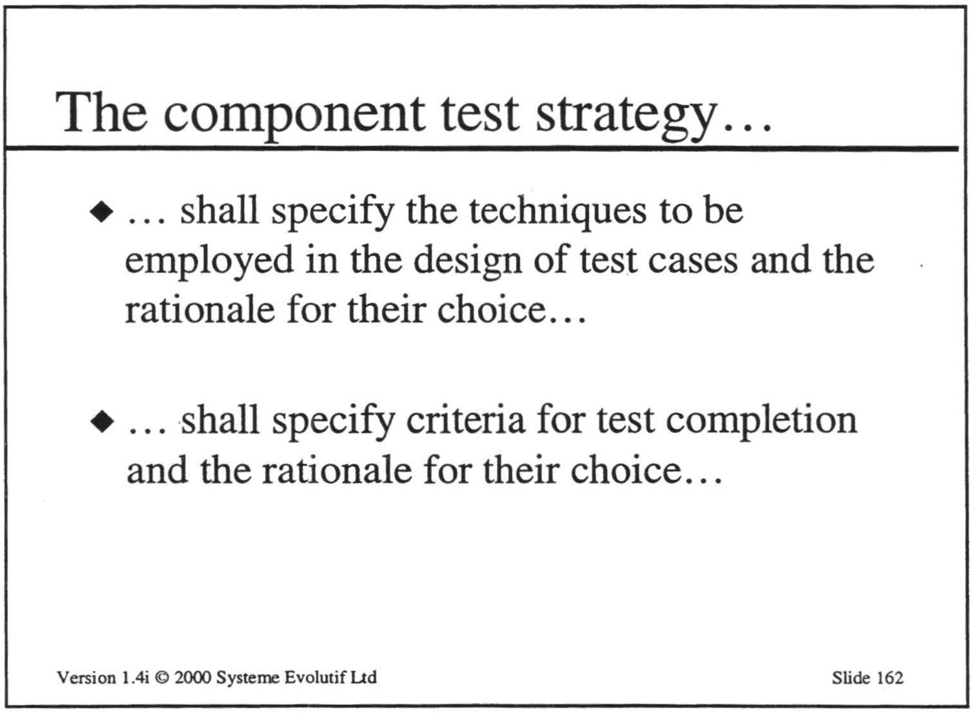
Slide 160



The standard does not cover…

- Selection of test design or measurement techniques
- Personnel selection or who does the testing
- Implementation (how required attributes of the test process are to be achieved e.g. tools)
- Fault removal (a separate process to fault detection).

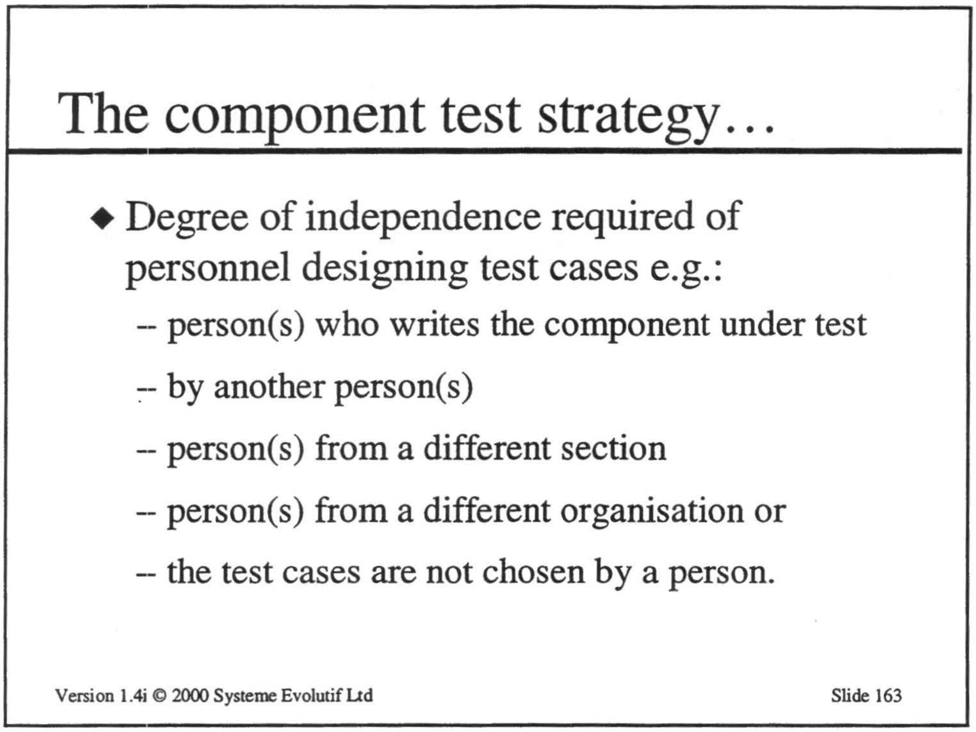
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 161



The component test strategy…

- … shall specify the techniques to be employed in the design of test cases and the rationale for their choice…
- … shall specify criteria for test completion and the rationale for their choice…

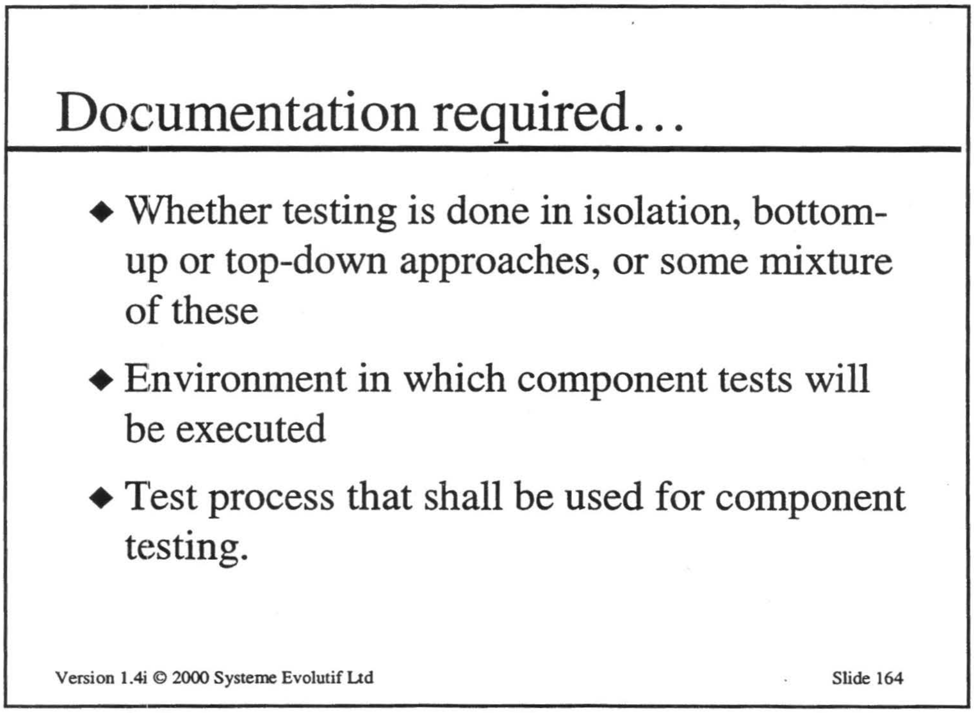
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 162



The component test strategy…

- Degree of independence required of personnel designing test cases e.g.:
  - -- person(s) who writes the component under test
  - -- by another person(s)
  - -- person(s) from a different section
  - -- person(s) from a different organisation or
  - -- the test cases are not chosen by a person.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 163

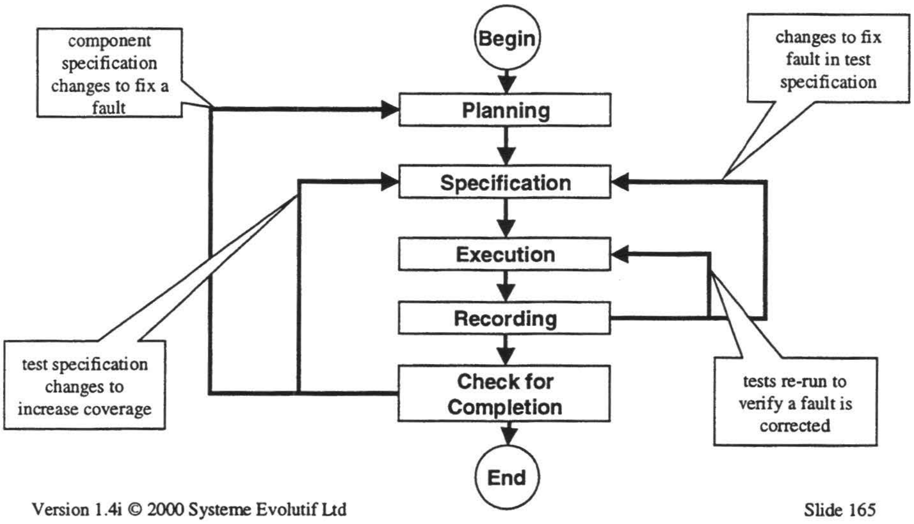


Documentation required…

- Whether testing is done in isolation, bottom-up or top-down approaches, or some mixture of these
- Environment in which component tests will be executed
- Test process that shall be used for component testing.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 164

Component test process



component specification changes to fix a fault

Begin

Planning

Specification

Execution

Recording

Check for Completion

End

changes to fix fault in test specification

test specification changes to increase coverage

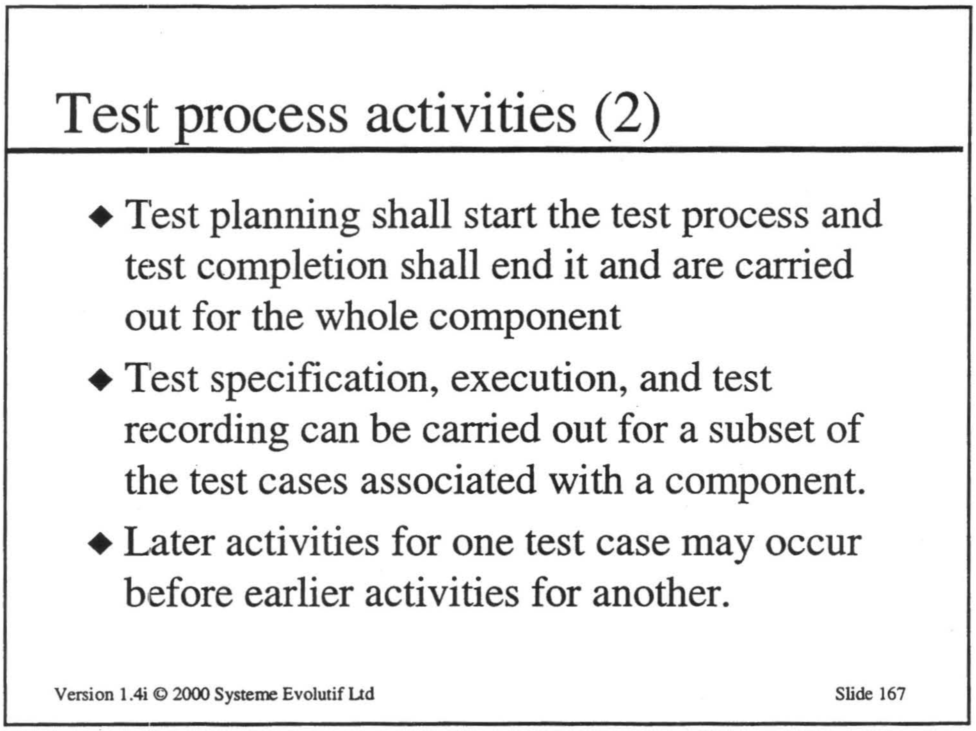
tests re-run to verify a fault is corrected

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 165

Test process activities

- Must occur in this order:
  - component test planning
  - component test specification
  - component test execution
  - component test recording
  - verification of component test completion.

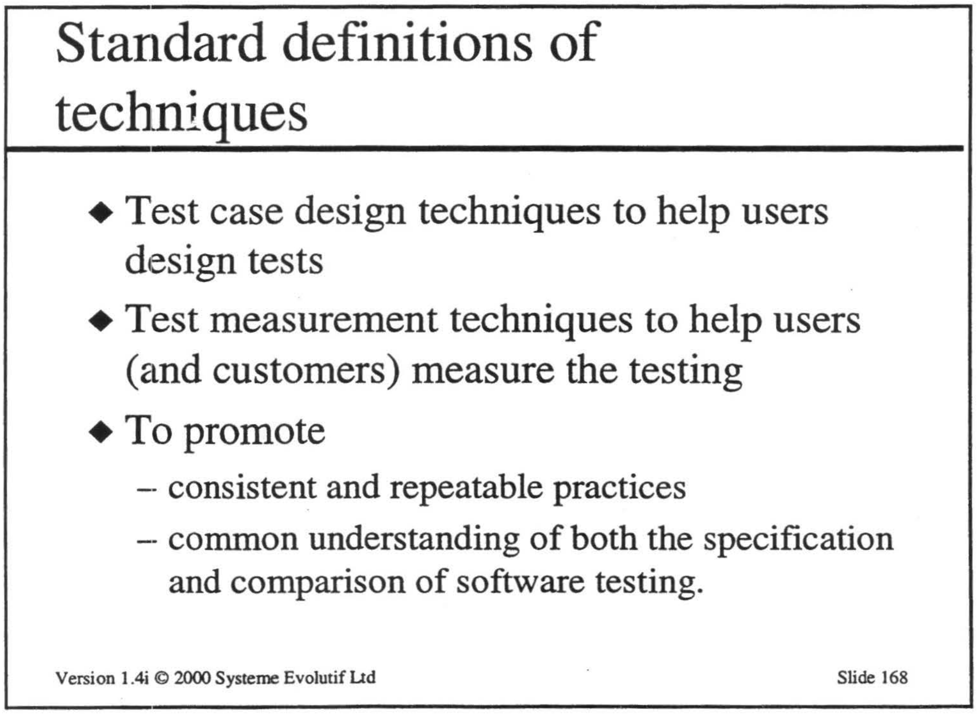
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 166



Test process activities (2)

- ◆ Test planning shall start the test process and test completion shall end it and are carried out for the whole component
- ◆ Test specification, execution, and test recording can be carried out for a subset of the test cases associated with a component.
- ◆ Later activities for one test case may occur before earlier activities for another.

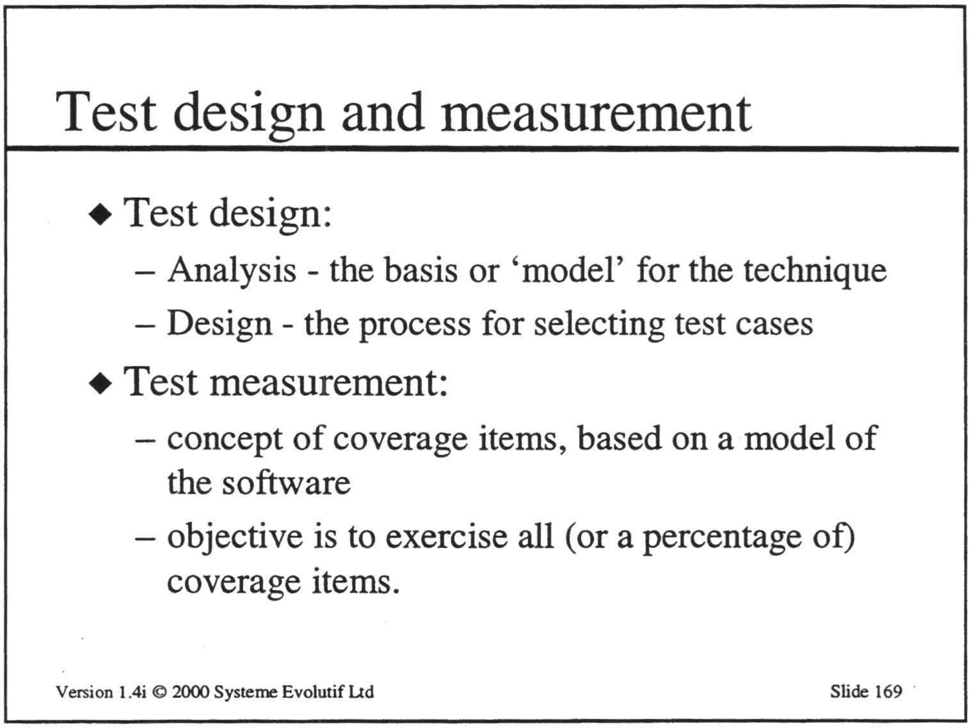
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 167



Standard definitions of techniques

- ◆ Test case design techniques to help users design tests
- ◆ Test measurement techniques to help users (and customers) measure the testing
- ◆ To promote
  - – consistent and repeatable practices
  - – common understanding of both the specification and comparison of software testing.

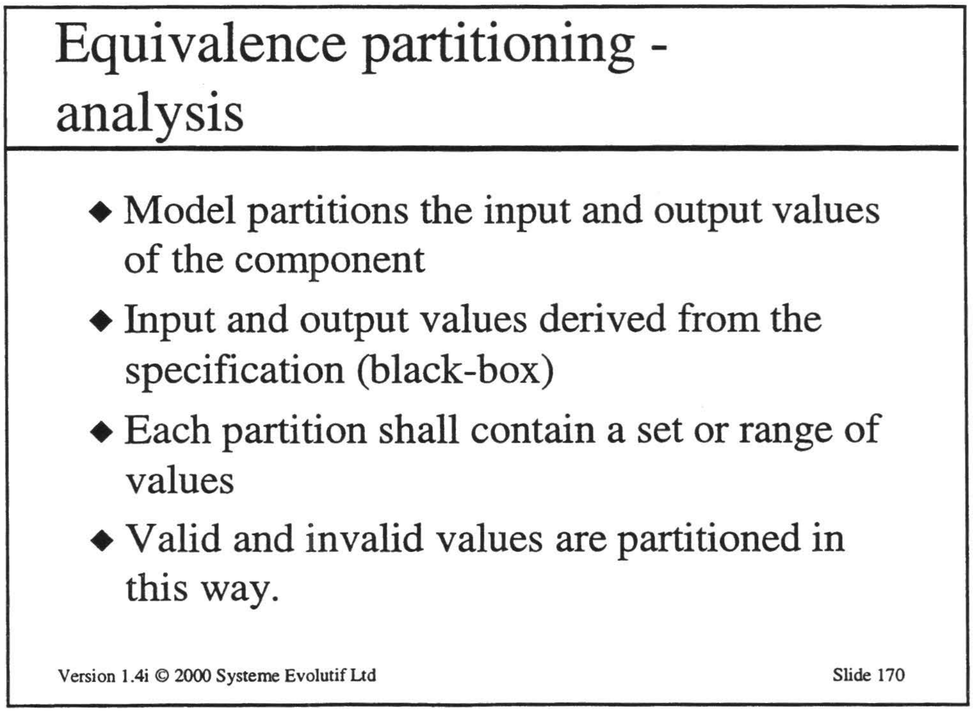
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 168



Test design and measurement

- Test design:
  - Analysis - the basis or 'model' for the technique
  - Design - the process for selecting test cases
- Test measurement:
  - concept of coverage items, based on a model of the software
  - objective is to exercise all (or a percentage of) coverage items.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 169



Equivalence partitioning - analysis

- Model partitions the input and output values of the component
- Input and output values derived from the specification (black-box)
- Each partition shall contain a set or range of values
- Valid and invalid values are partitioned in this way.

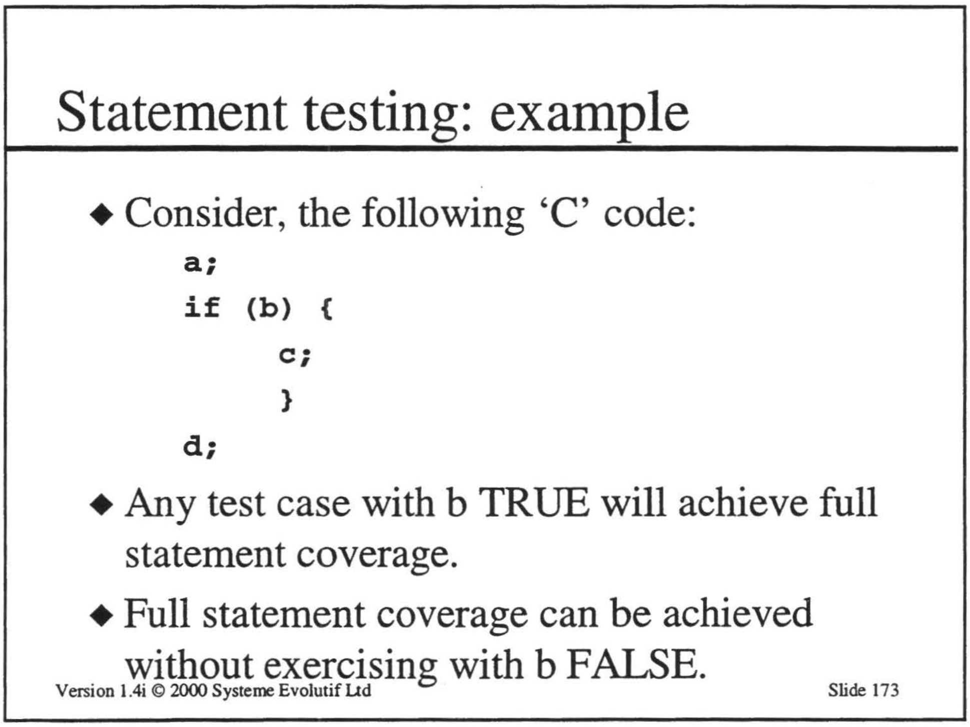
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 170

Equivalence partitioning - design

- Test cases shall be designed to exercise partitions
- A test case shall comprise the following:
  - the input(s)
  - the partitions exercised
  - the expected outcome of the test case.
- Test cases are designed to exercise partitions of valid values, and invalid input values.

Statement testing - coverage items

- Coverage item is an *executable statement*
  - a statement should be an atomic action, executed completely or not at all
- For example:
  - assignments, decisions
  - procedure and function calls
  - variable declarations with explicit initialisations
  - dynamic allocation of variable storage on a heap.



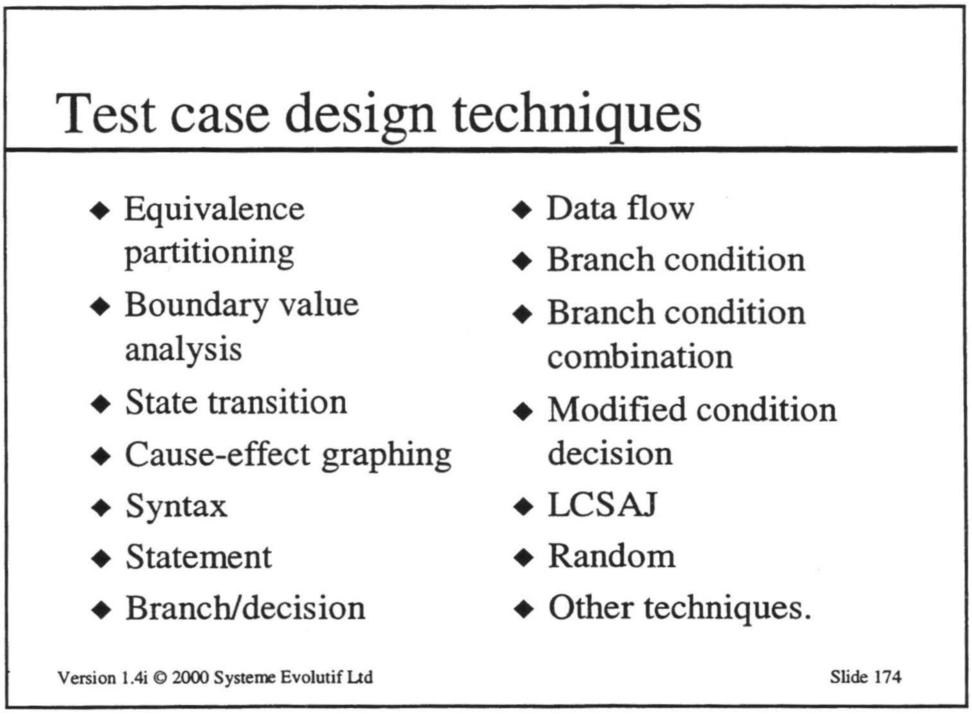
Statement testing: example

- Consider, the following ‘C’ code:

```
a;
if (b) {
     c;
     }
d;
```

- Any test case with b TRUE will achieve full statement coverage.
- Full statement coverage can be achieved without exercising with b FALSE.

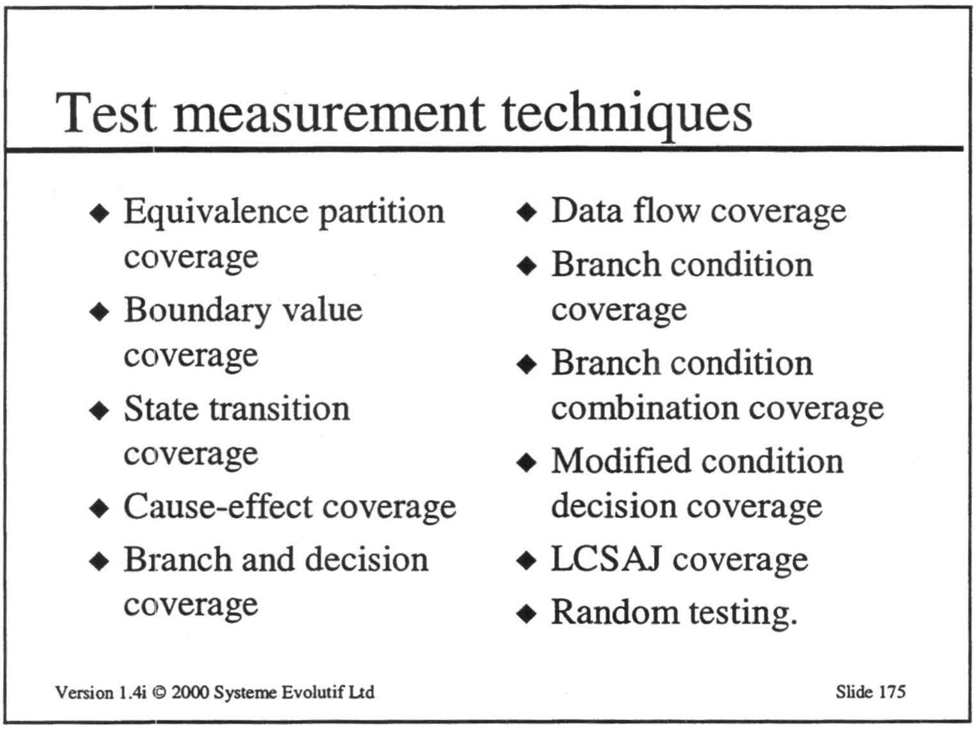
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 173



Test case design techniques

- Equivalence partitioning
- Boundary value analysis
- State transition
- Cause-effect graphing
- Syntax
- Statement
- Branch/decision
- Data flow
- Branch condition
- Branch condition combination
- Modified condition decision
- LCSAJ
- Random
- Other techniques.

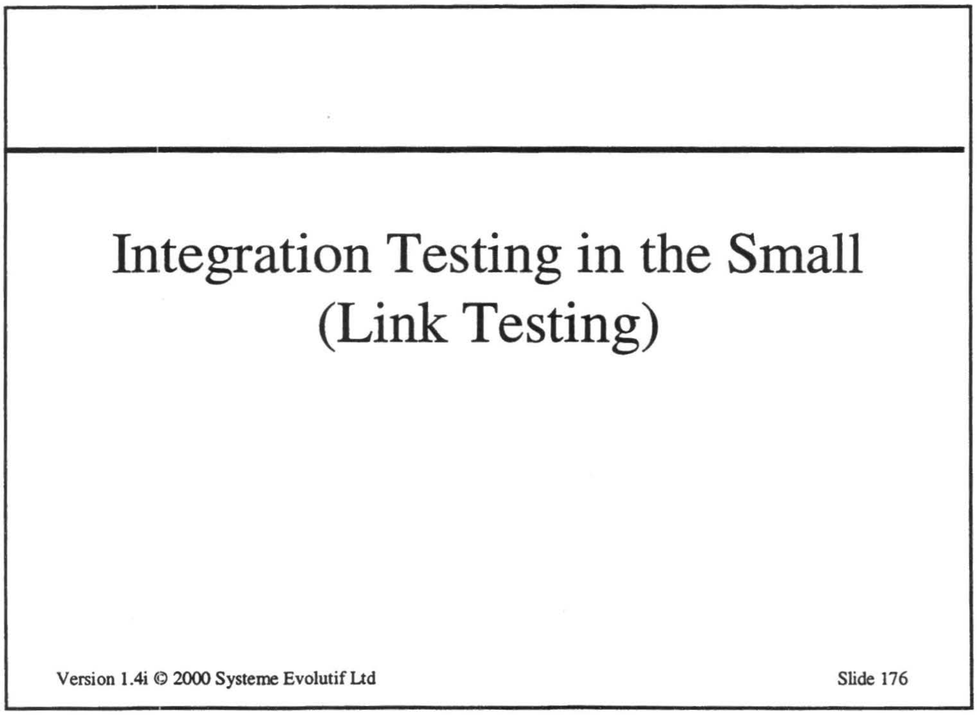
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 174



Test measurement techniques

- Equivalence partition coverage
- Boundary value coverage
- State transition coverage
- Cause-effect coverage
- Branch and decision coverage
- Data flow coverage
- Branch condition coverage
- Branch condition combination coverage
- Modified condition decision coverage
- LCSAJ coverage
- Random testing.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 175



Integration Testing in the Small (Link Testing)

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 176

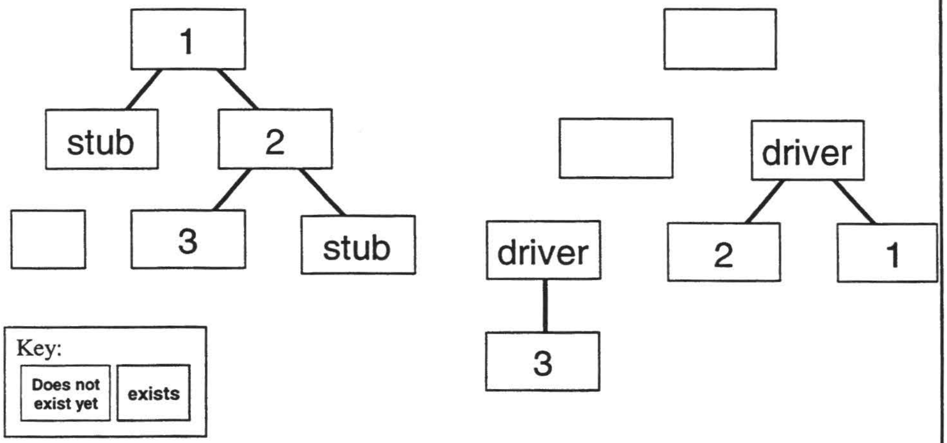
Software integration and link testing

- ◆ In the coding stage, you are performing "integration in the very small"
- ◆ Strategies for coding and integration:
  - – bottom up, top down, "big bang"
  - – appropriate in different situations
- ◆ Choice based on programming tool
- ◆ Testing also affects choice of integration strategy.

Version 1.4i © 2000 Systeme Evolutif Ltd

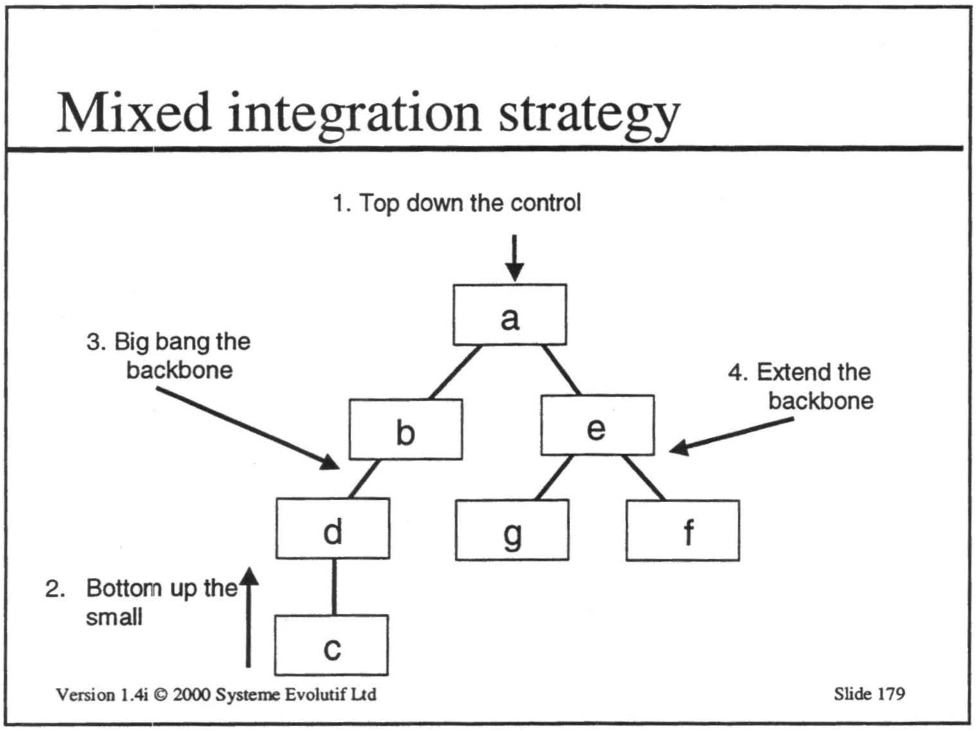
Slide 177

Stubs and drivers



Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 178



Mixed integration strategy

1. Top down the control

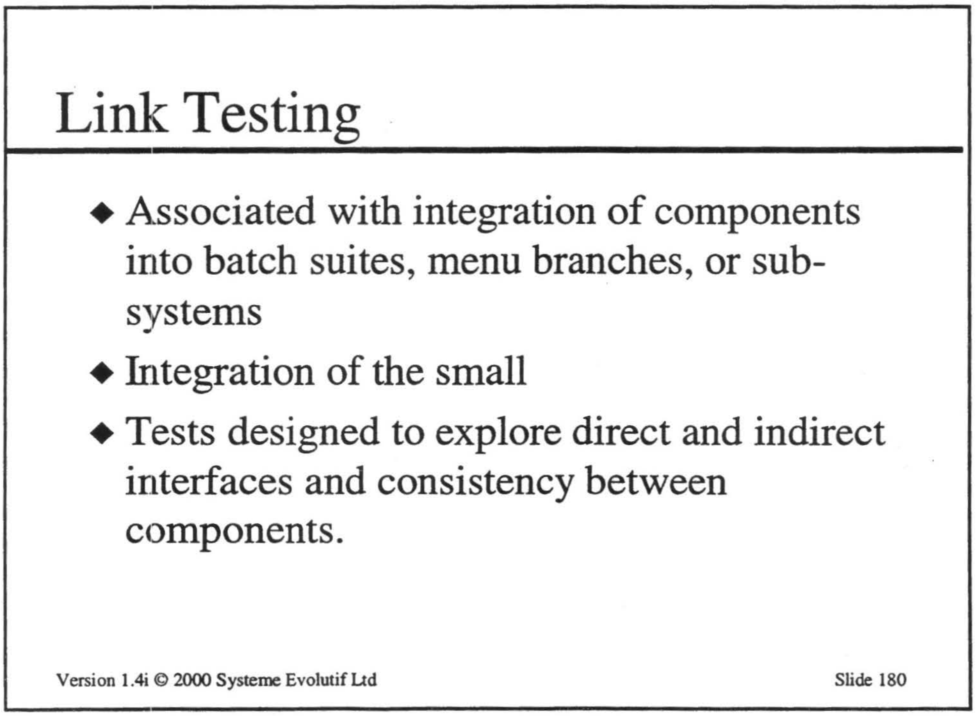
3. Big bang the backbone

4. Extend the backbone

2. Bottom up the small

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 179

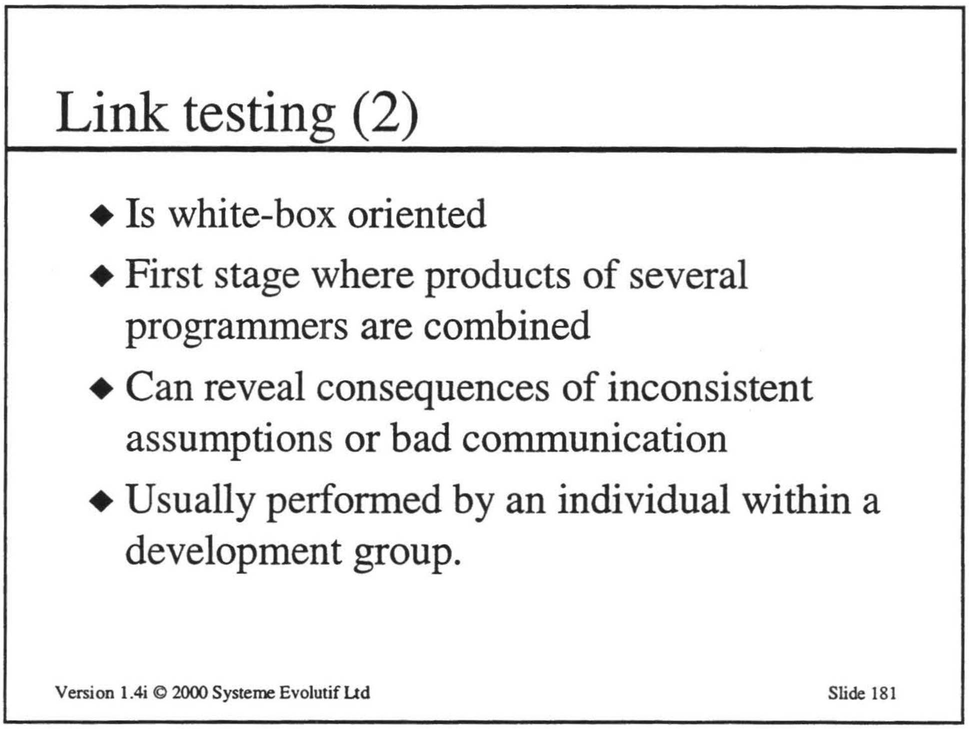


Link Testing

- Associated with integration of components into batch suites, menu branches, or sub-systems
- Integration of the small
- Tests designed to explore direct and indirect interfaces and consistency between components.

Version 1.4i © 2000 Systeme Evolutif Ltd

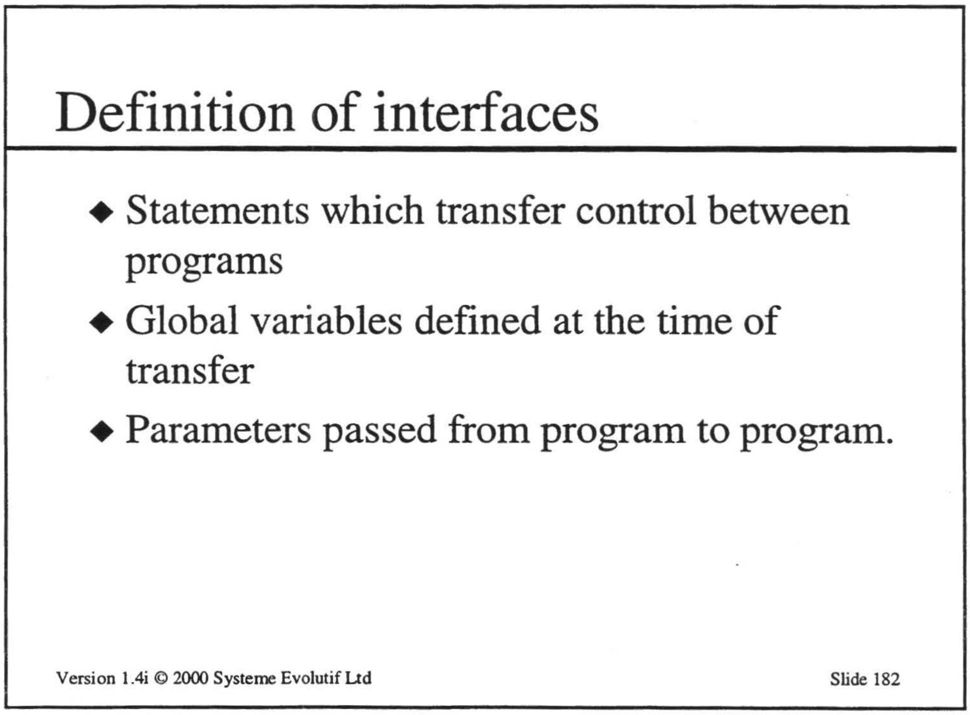
Slide 180



Link testing (2)

- Is white-box oriented
- First stage where products of several programmers are combined
- Can reveal consequences of inconsistent assumptions or bad communication
- Usually performed by an individual within a development group.

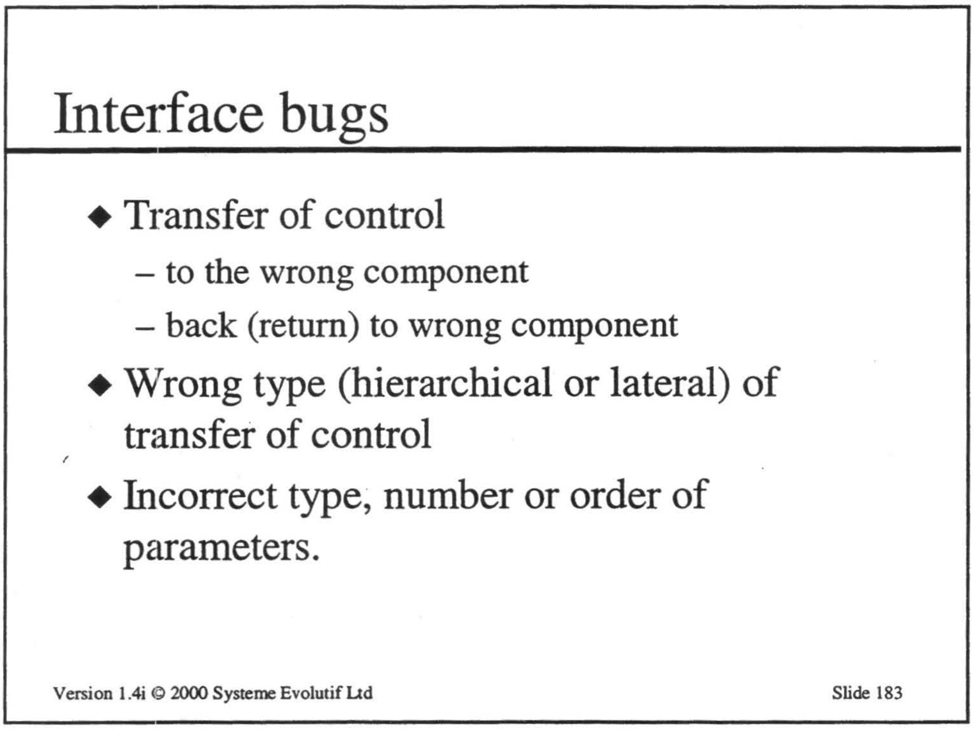
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 181



Definition of interfaces

- Statements which transfer control between programs
- Global variables defined at the time of transfer
- Parameters passed from program to program.

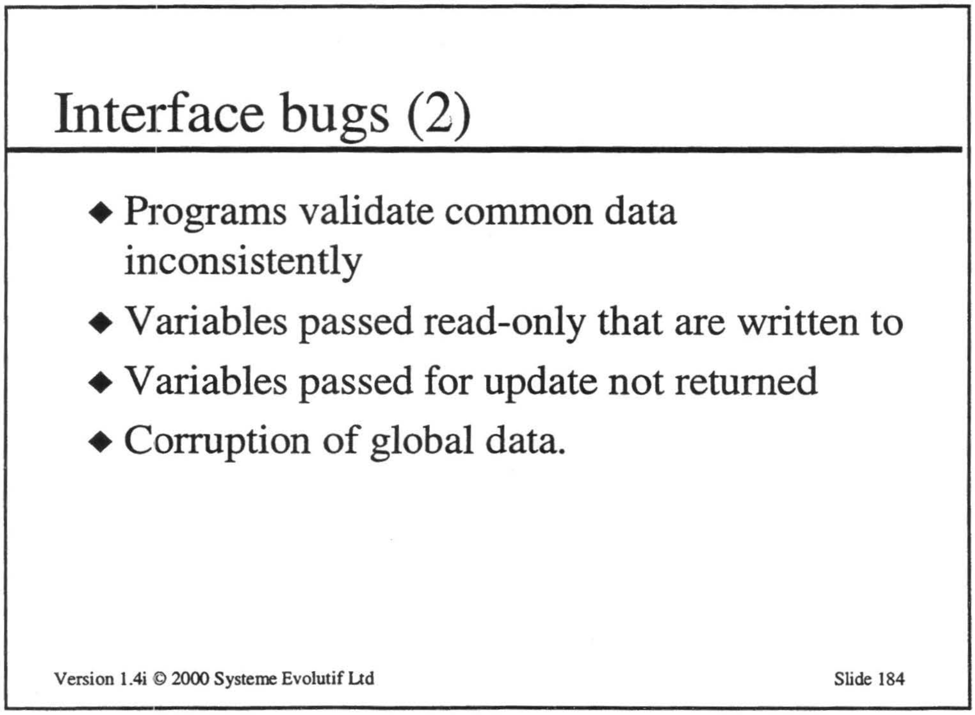
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 182



Interface bugs

- Transfer of control
  - to the wrong component
  - back (return) to wrong component
- Wrong type (hierarchical or lateral) of transfer of control
- Incorrect type, number or order of parameters.

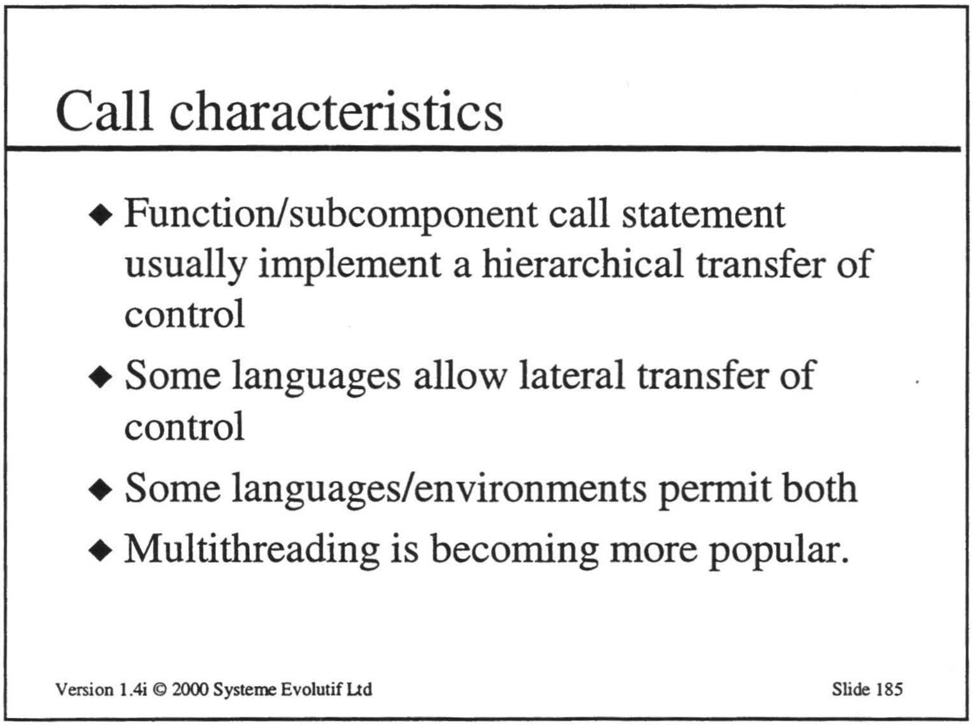
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 183



Interface bugs (2)

- Programs validate common data inconsistently
- Variables passed read-only that are written to
- Variables passed for update not returned
- Corruption of global data.

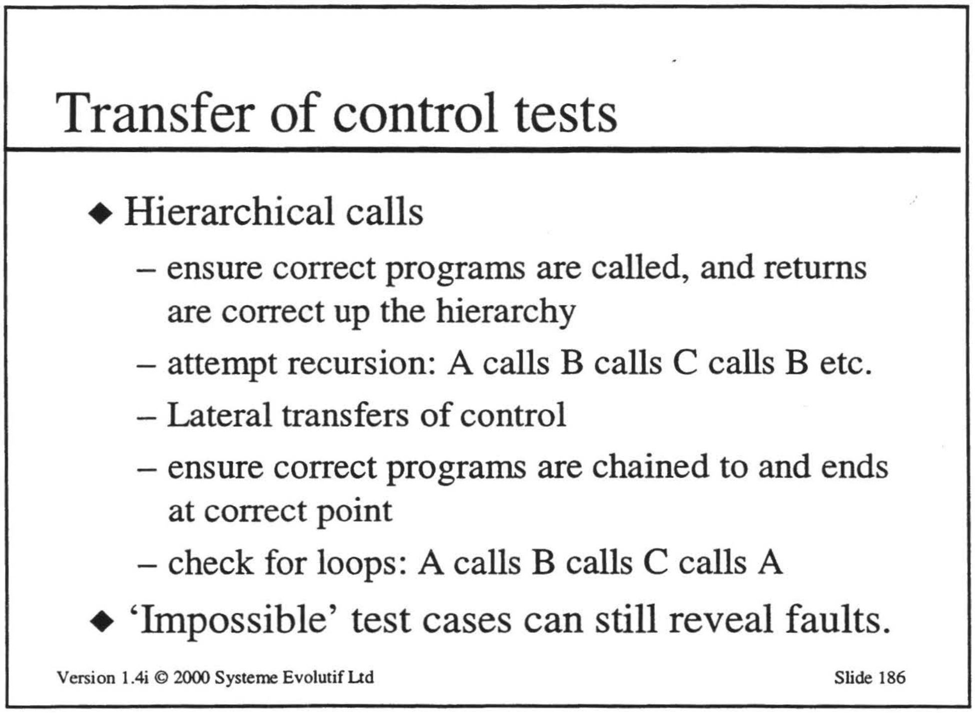
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 184



Call characteristics

- Function/subcomponent call statement usually implement a hierarchical transfer of control
- Some languages allow lateral transfer of control
- Some languages/environments permit both
- Multithreading is becoming more popular.

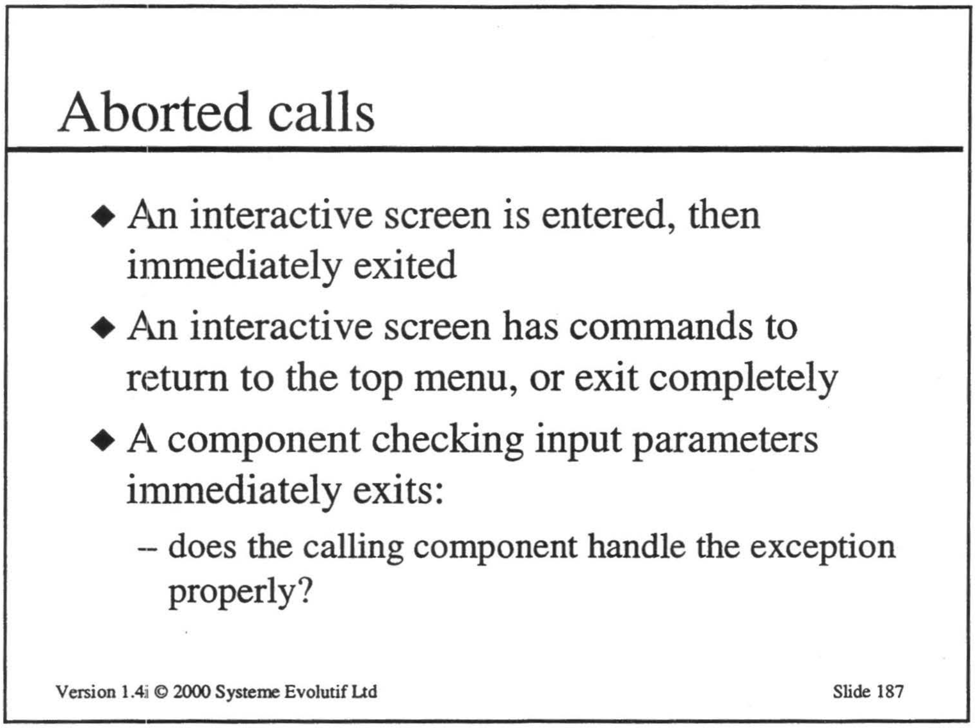
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 185



Transfer of control tests

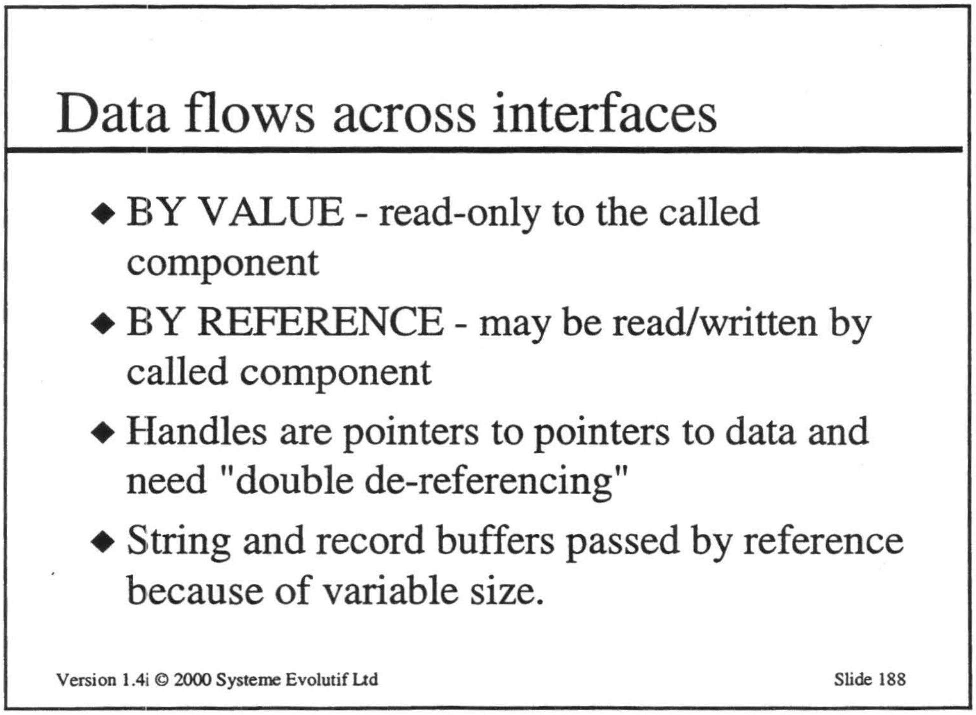
- Hierarchical calls
  - ensure correct programs are called, and returns are correct up the hierarchy
  - attempt recursion: A calls B calls C calls B etc.
  - Lateral transfers of control
  - ensure correct programs are chained to and ends at correct point
  - check for loops: A calls B calls C calls A
- 'Impossible' test cases can still reveal faults.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 186



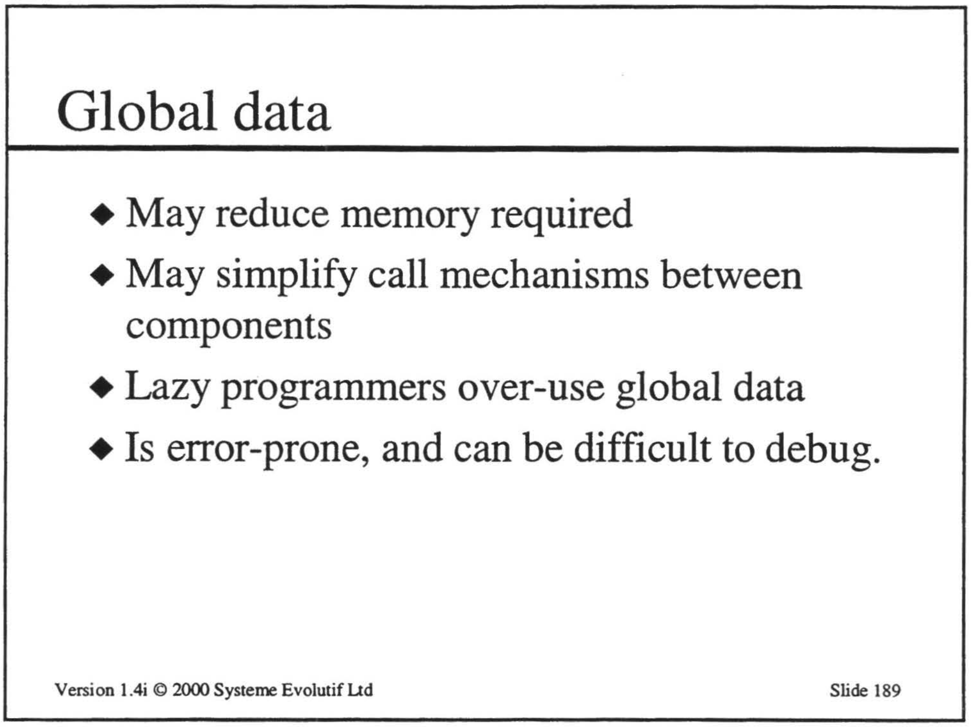
Aborted calls

- An interactive screen is entered, then immediately exited
- An interactive screen has commands to return to the top menu, or exit completely
- A component checking input parameters immediately exits:
  - does the calling component handle the exception properly?



Data flows across interfaces

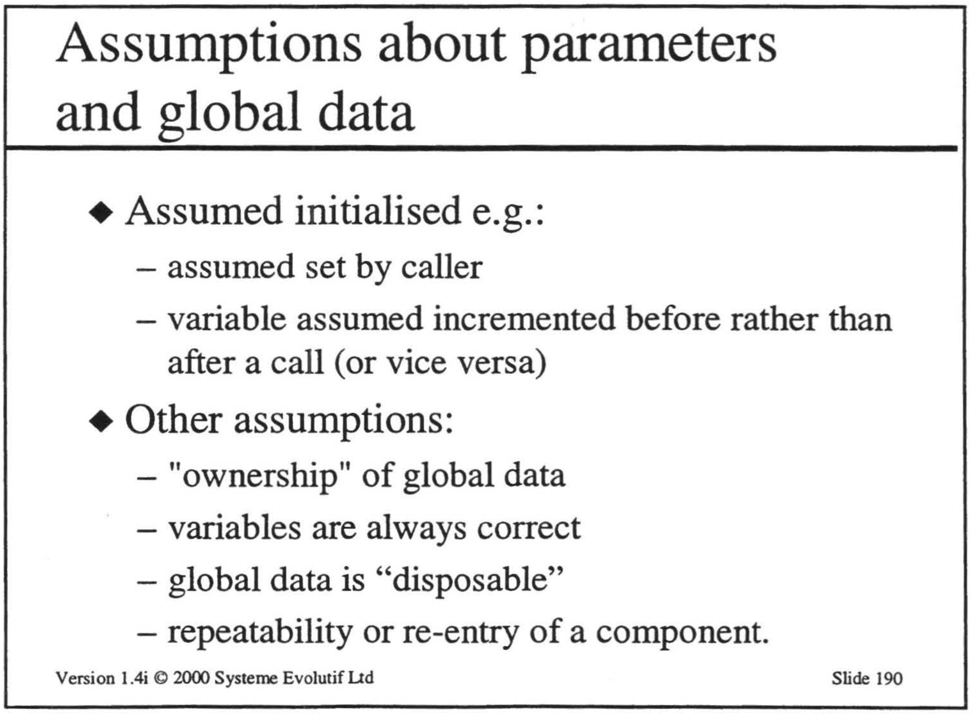
- BY VALUE - read-only to the called component
- BY REFERENCE - may be read/written by called component
- Handles are pointers to pointers to data and need "double de-referencing"
- String and record buffers passed by reference because of variable size.



Global data

- May reduce memory required
- May simplify call mechanisms between components
- Lazy programmers over-use global data
- Is error-prone, and can be difficult to debug.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 189



Assumptions about parameters and global data

- Assumed initialised e.g.:
  - – assumed set by caller
  - – variable assumed incremented before rather than after a call (or vice versa)
- Other assumptions:
  - – "ownership" of global data
  - – variables are always correct
  - – global data is “disposable”
  - – repeatability or re-entry of a component.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 190

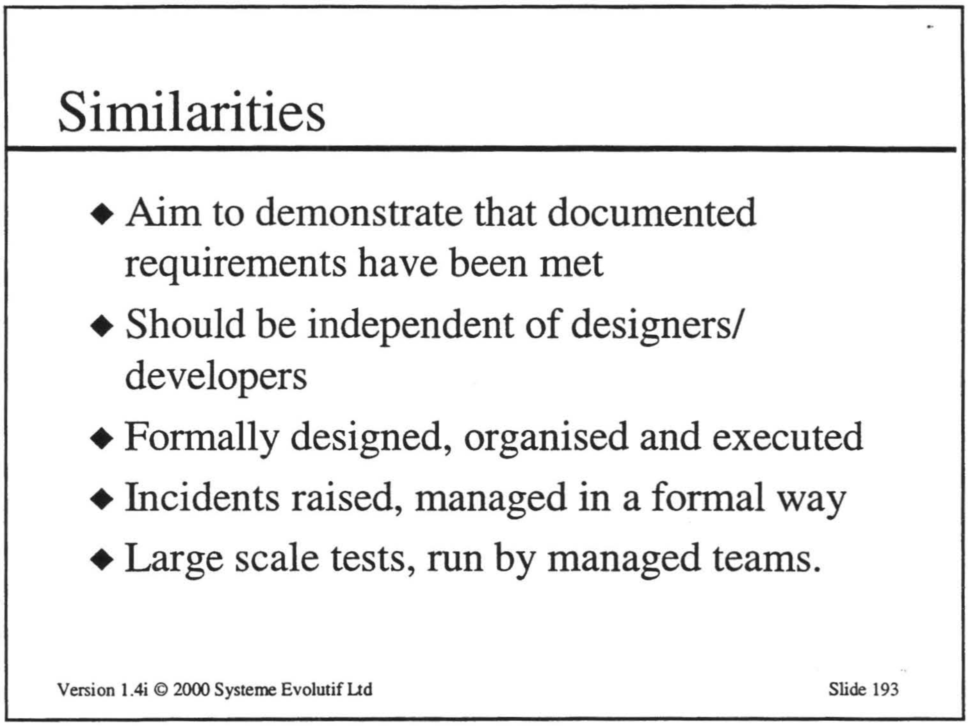
Inter-component parameter checking

- Does the called component explicitly check input parameters?
- Does the calling component check
  - called component return status?
  - returned values?
- Programming or interface standards should define whether callers, called or both components perform checking and under what circumstances.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 191

System and Acceptance Testing

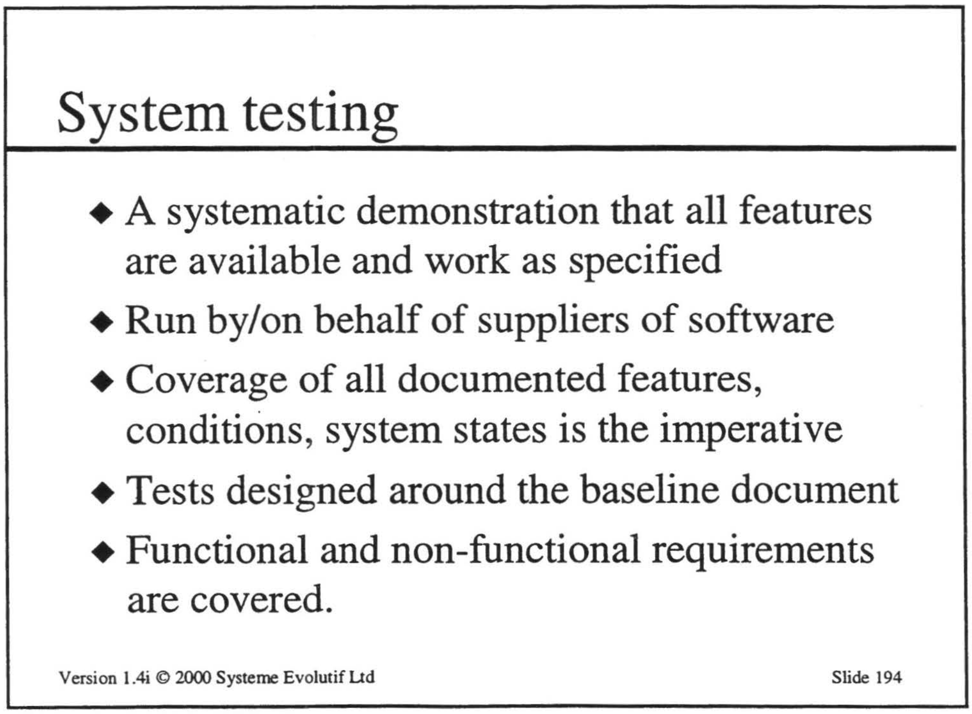
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 192



Similarities

- Aim to demonstrate that documented requirements have been met
- Should be independent of designers/ developers
- Formally designed, organised and executed
- Incidents raised, managed in a formal way
- Large scale tests, run by managed teams.

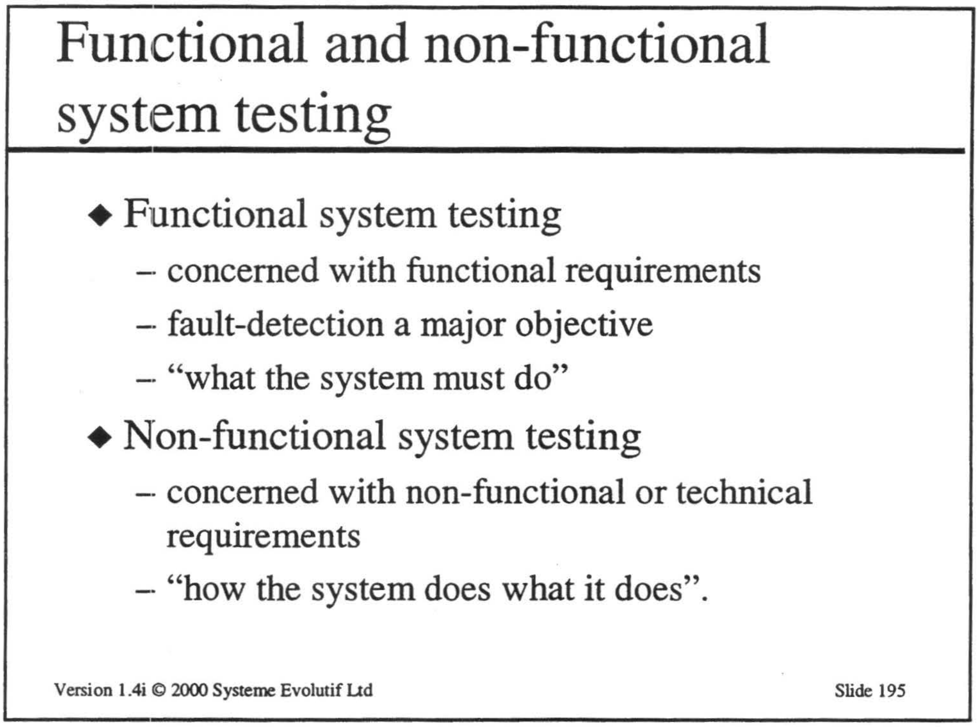
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 193



System testing

- A systematic demonstration that all features are available and work as specified
- Run by/on behalf of suppliers of software
- Coverage of all documented features, conditions, system states is the imperative
- Tests designed around the baseline document
- Functional and non-functional requirements are covered.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 194

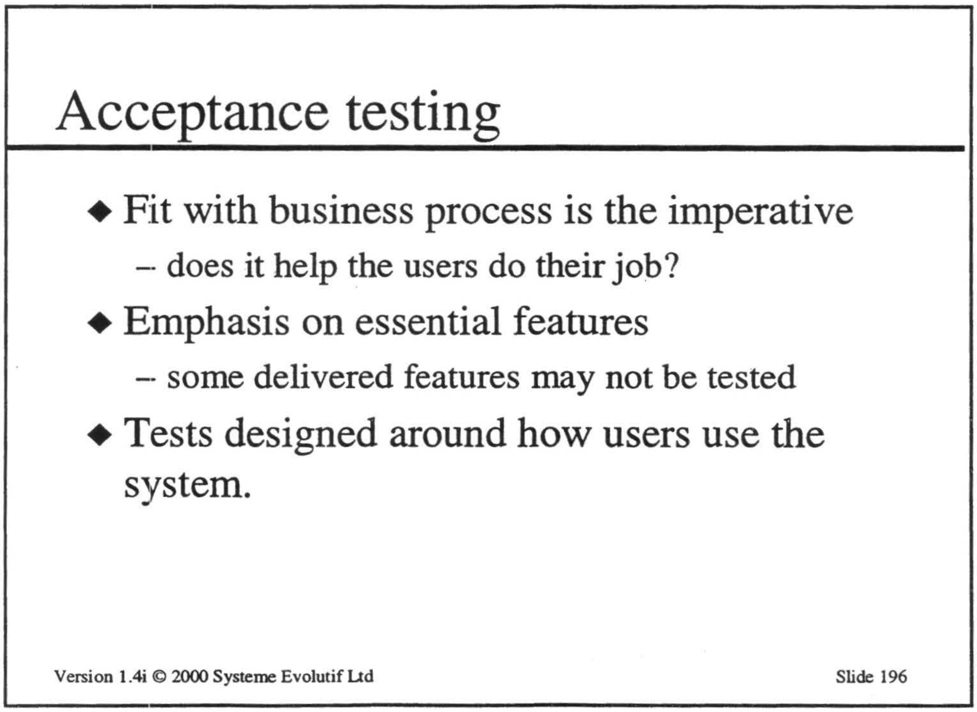


Functional and non-functional system testing

- Functional system testing
  - concerned with functional requirements
  - fault-detection a major objective
  - "what the system must do"
- Non-functional system testing
  - concerned with non-functional or technical requirements
  - "how the system does what it does".

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 195

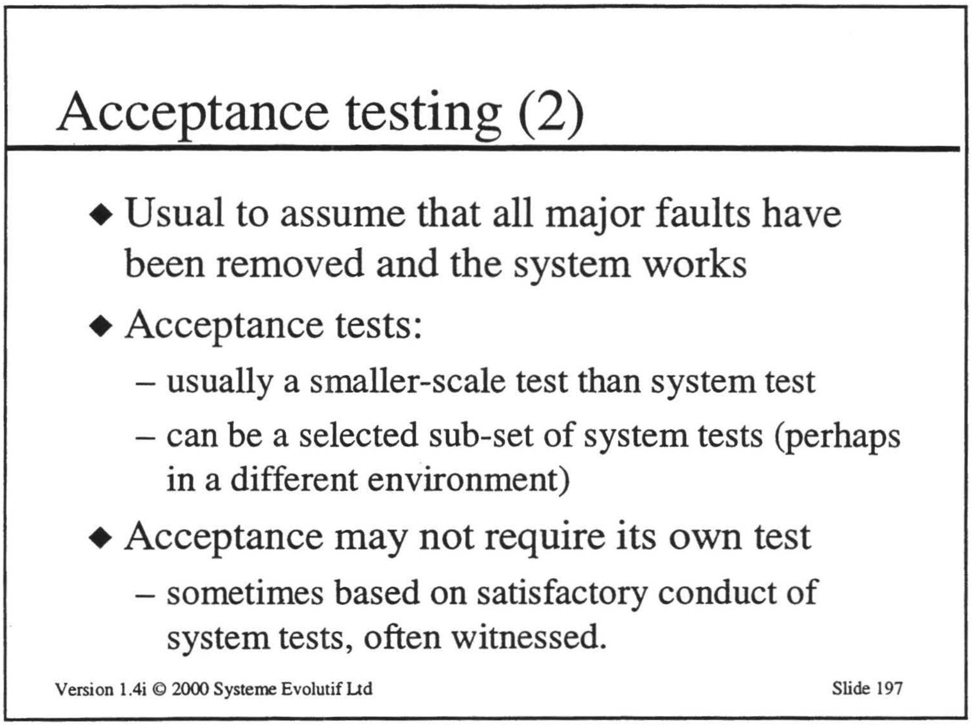


Acceptance testing

- Fit with business process is the imperative
  - does it help the users do their job?
- Emphasis on essential features
  - some delivered features may not be tested
- Tests designed around how users use the system.

Version 1.4i © 2000 Systeme Evolutif Ltd

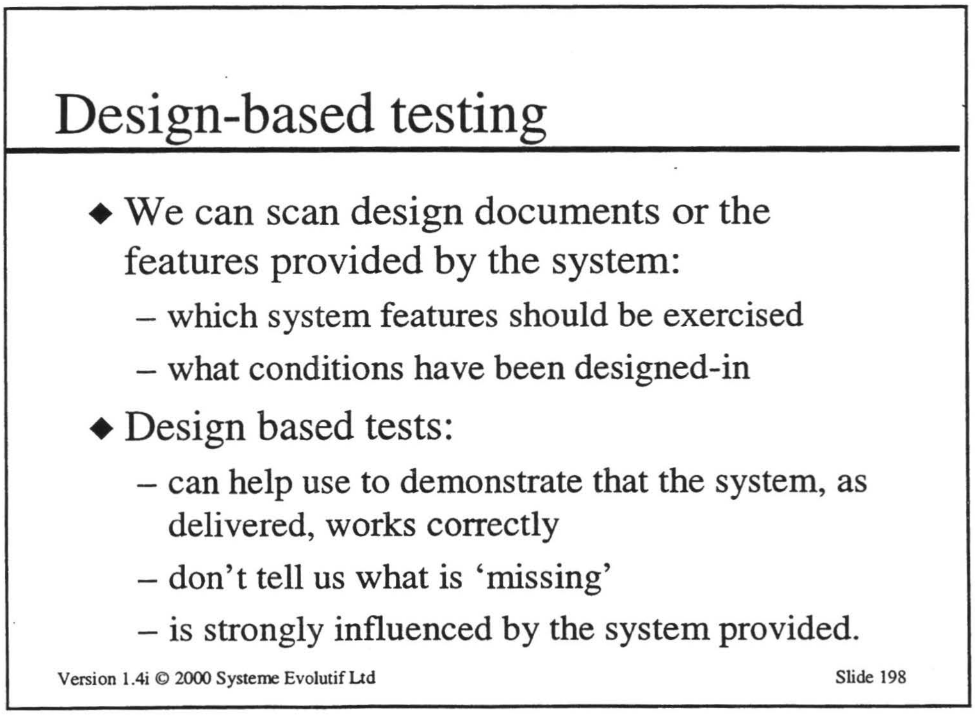
Slide 196



Acceptance testing (2)

- Usual to assume that all major faults have been removed and the system works
- Acceptance tests:
  - usually a smaller-scale test than system test
  - can be a selected sub-set of system tests (perhaps in a different environment)
- Acceptance may not require its own test
  - sometimes based on satisfactory conduct of system tests, often witnessed.

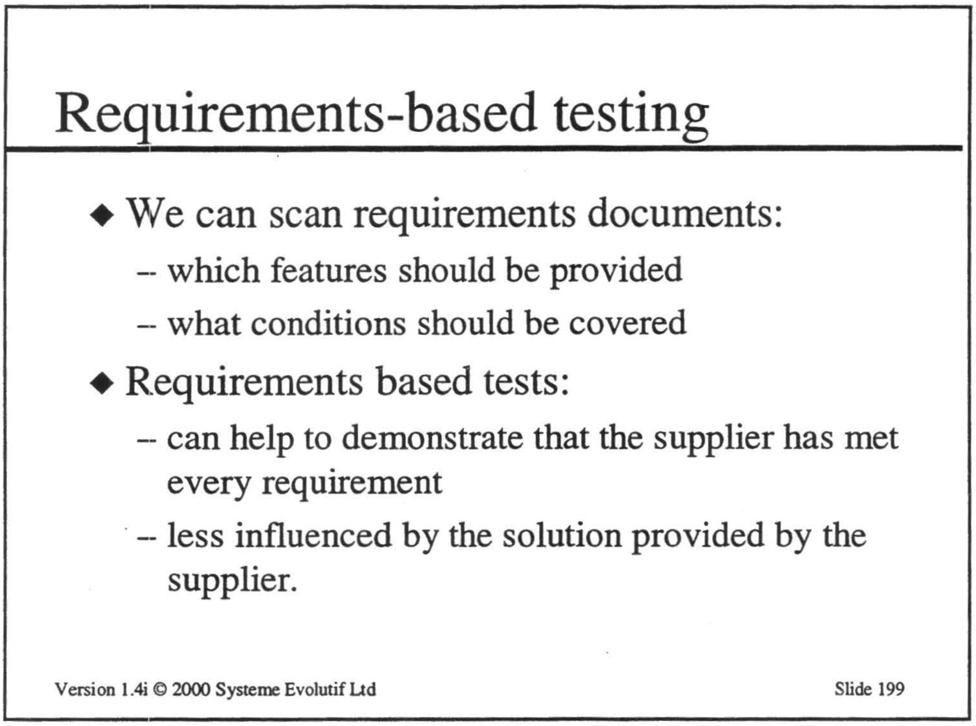
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 197



Design-based testing

- We can scan design documents or the features provided by the system:
  - which system features should be exercised
  - what conditions have been designed-in
- Design based tests:
  - can help use to demonstrate that the system, as delivered, works correctly
  - don't tell us what is 'missing'
  - is strongly influenced by the system provided.

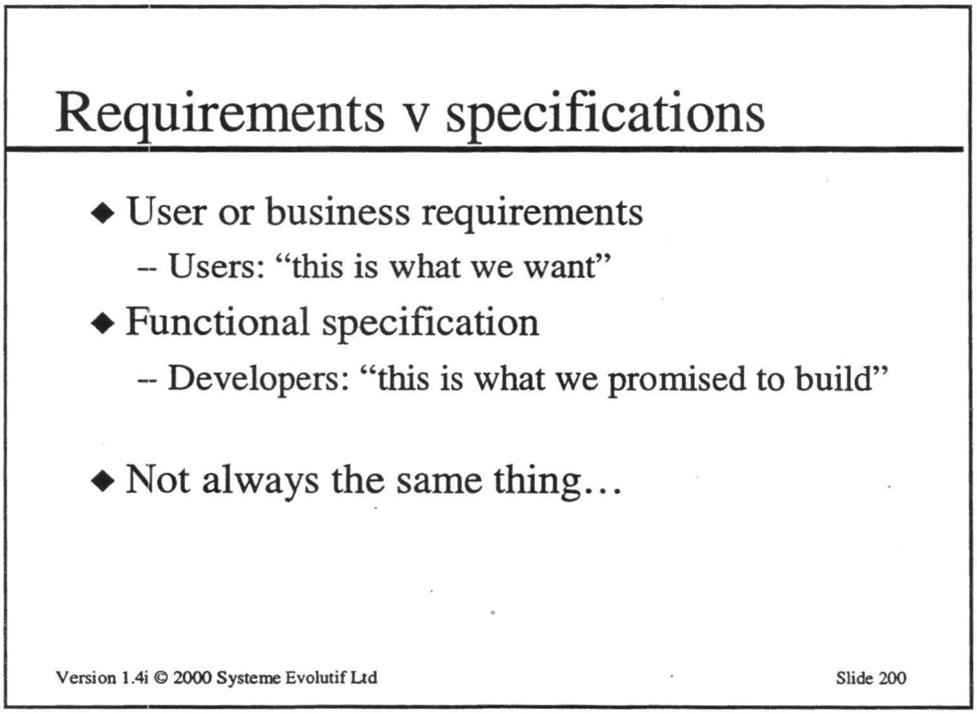
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 198



Requirements-based testing

- We can scan requirements documents:
  - which features should be provided
  - what conditions should be covered
- Requirements based tests:
  - can help to demonstrate that the supplier has met every requirement
  - less influenced by the solution provided by the supplier.

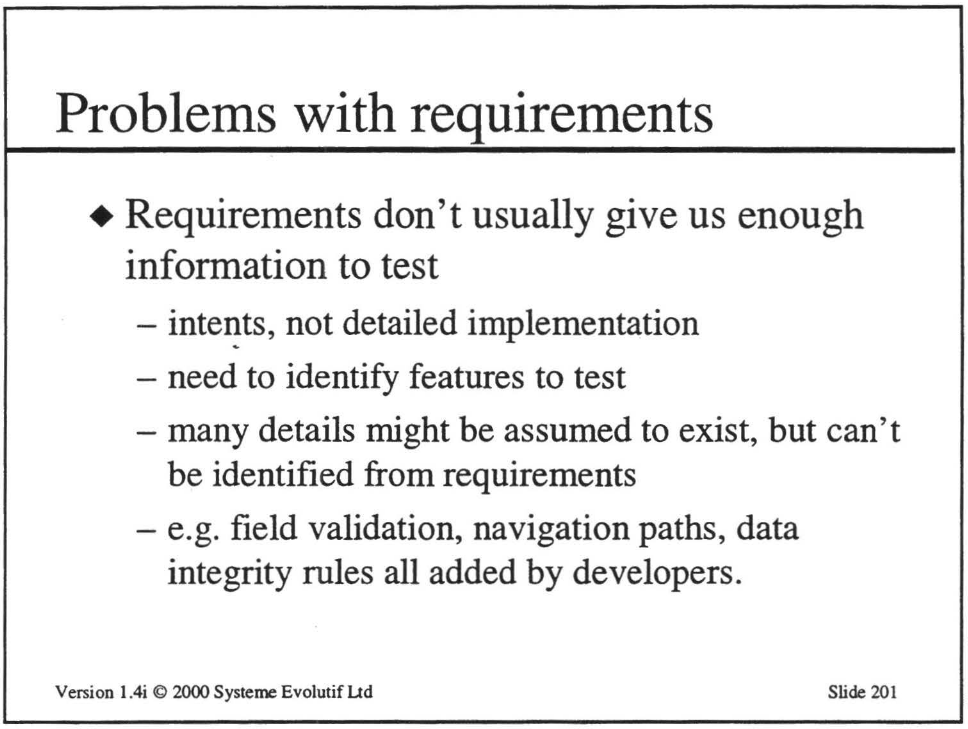
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 199



Requirements v specifications

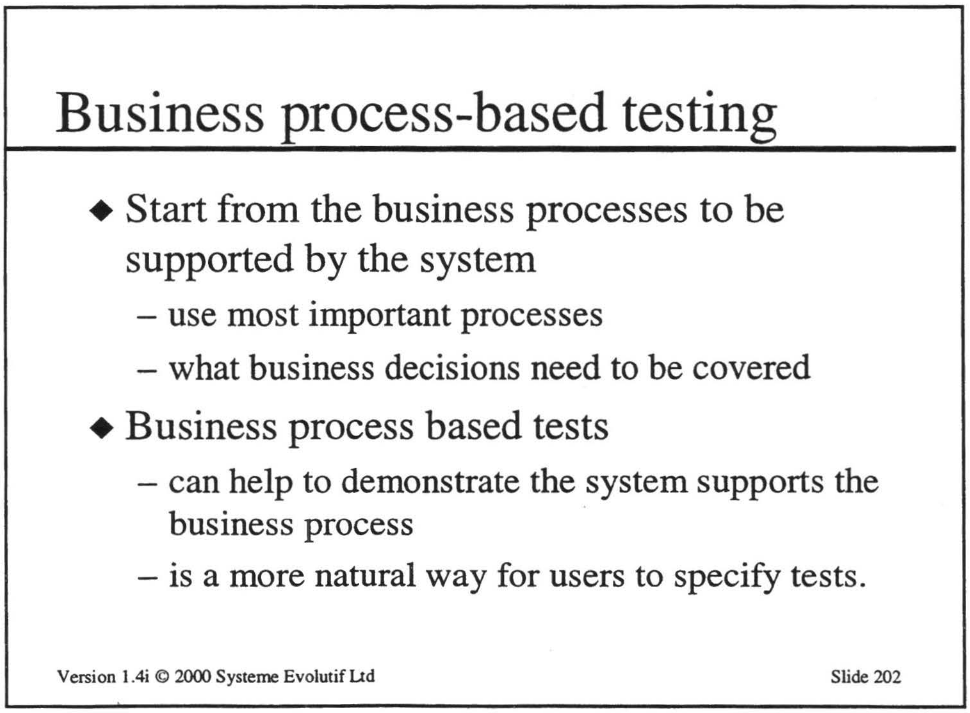
- User or business requirements
  - Users: “this is what we want”
- Functional specification
  - Developers: “this is what we promised to build”
- Not always the same thing…

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 200



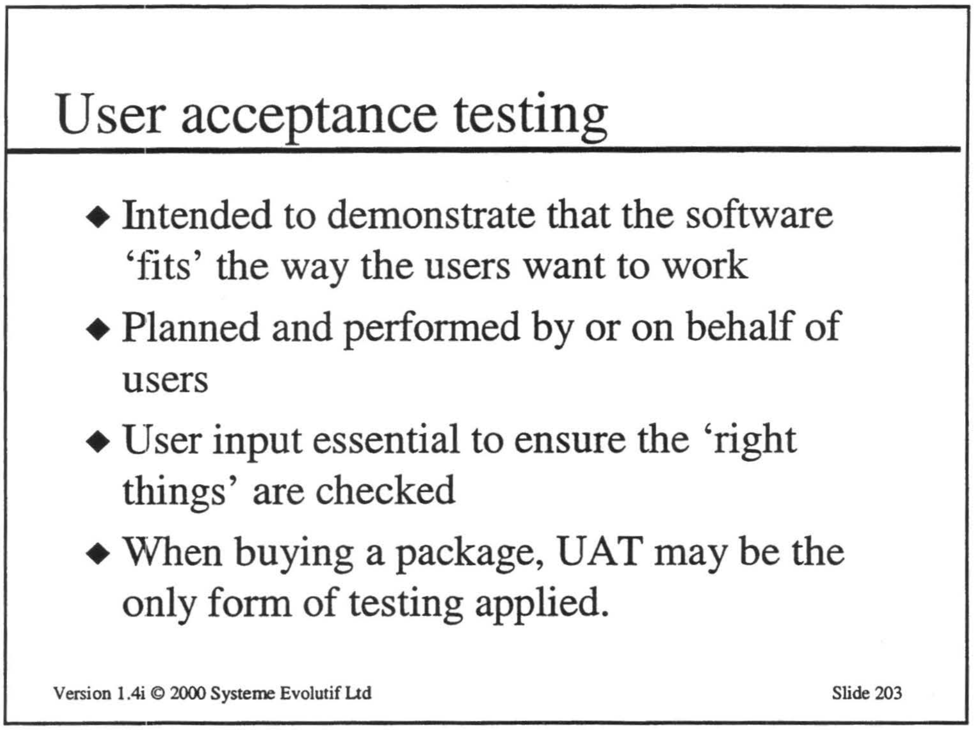
Problems with requirements

- ◆ Requirements don't usually give us enough information to test
  - – intents, not detailed implementation
  - – need to identify features to test
  - – many details might be assumed to exist, but can't be identified from requirements
  - – e.g. field validation, navigation paths, data integrity rules all added by developers.



Business process-based testing

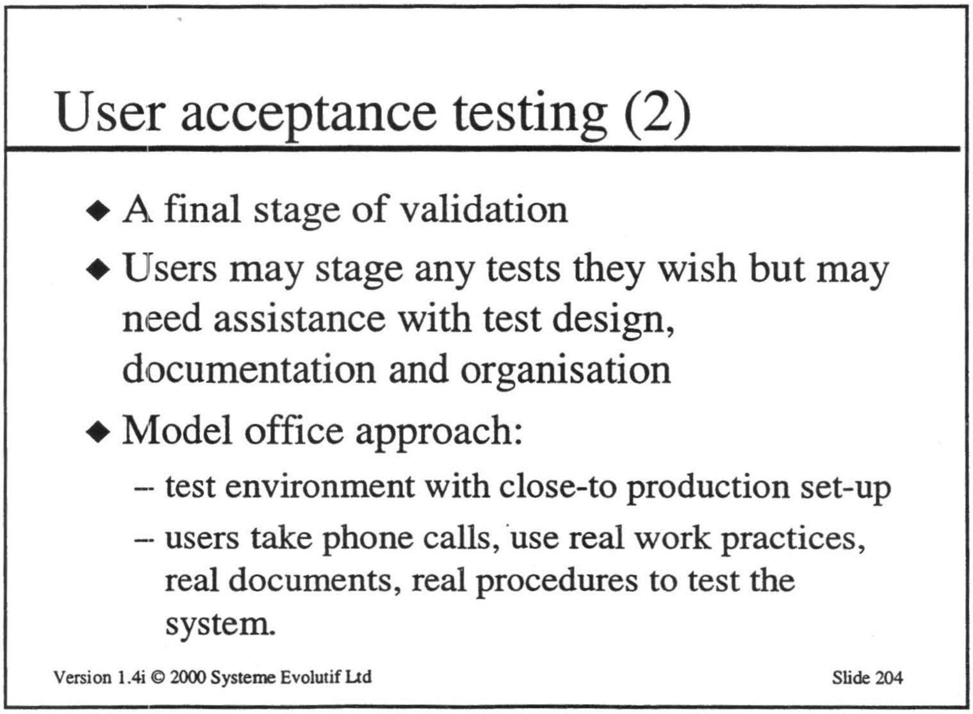
- ◆ Start from the business processes to be supported by the system
  - – use most important processes
  - – what business decisions need to be covered
- ◆ Business process based tests
  - – can help to demonstrate the system supports the business process
  - – is a more natural way for users to specify tests.



User acceptance testing

- Intended to demonstrate that the software ‘fits’ the way the users want to work
- Planned and performed by or on behalf of users
- User input essential to ensure the ‘right things’ are checked
- When buying a package, UAT may be the only form of testing applied.

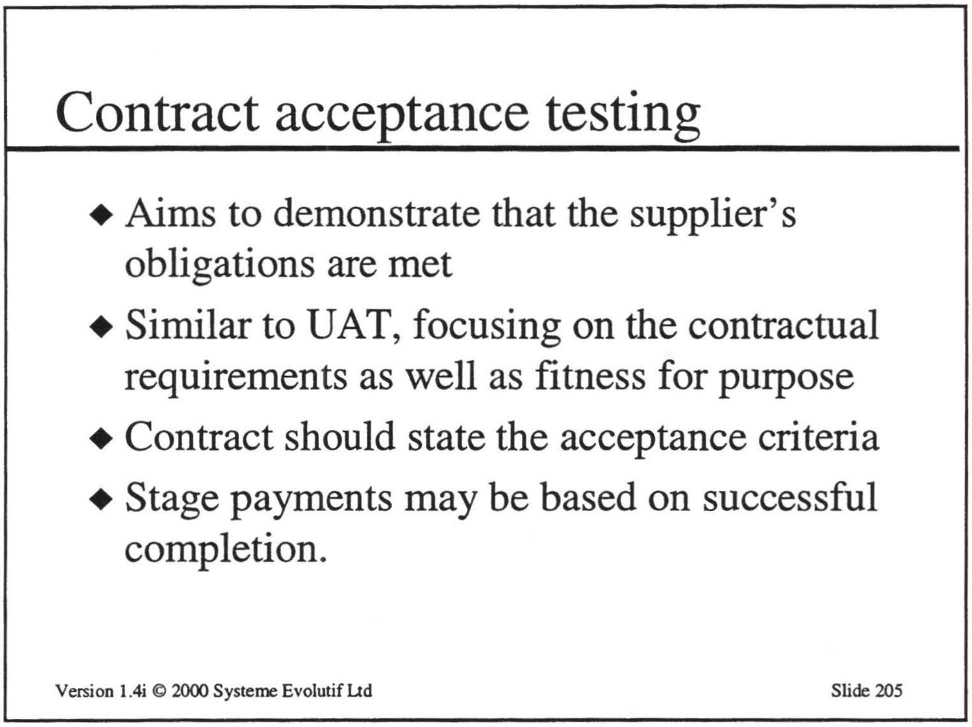
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 203



User acceptance testing (2)

- A final stage of validation
- Users may stage any tests they wish but may need assistance with test design, documentation and organisation
- Model office approach:
  - test environment with close-to production set-up
  - users take phone calls, use real work practices, real documents, real procedures to test the system.

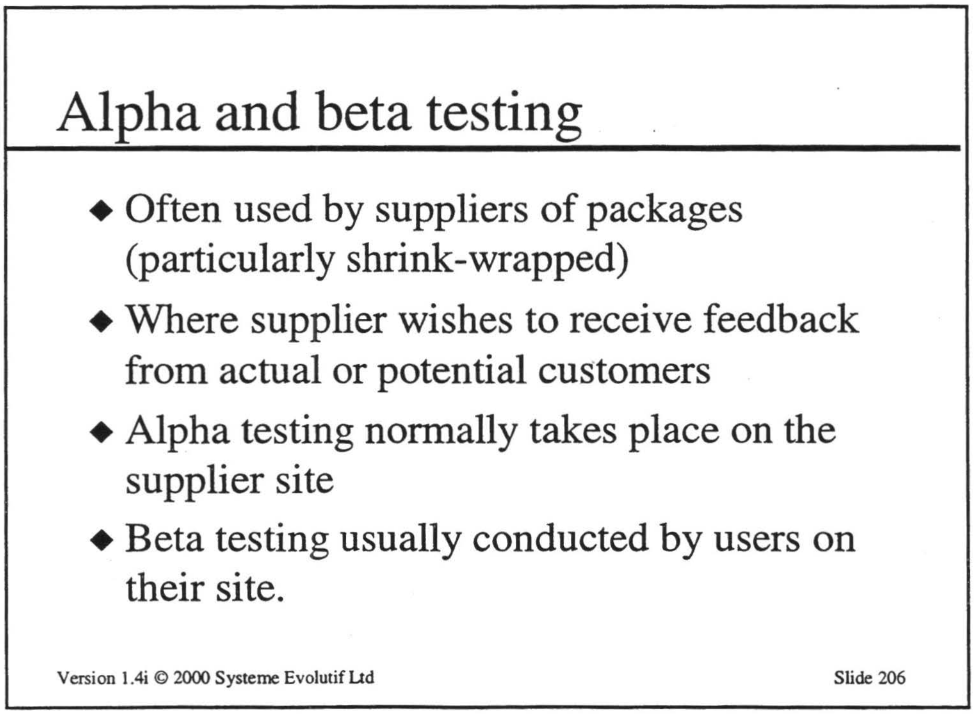
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 204



Contract acceptance testing

- Aims to demonstrate that the supplier's obligations are met
- Similar to UAT, focusing on the contractual requirements as well as fitness for purpose
- Contract should state the acceptance criteria
- Stage payments may be based on successful completion.

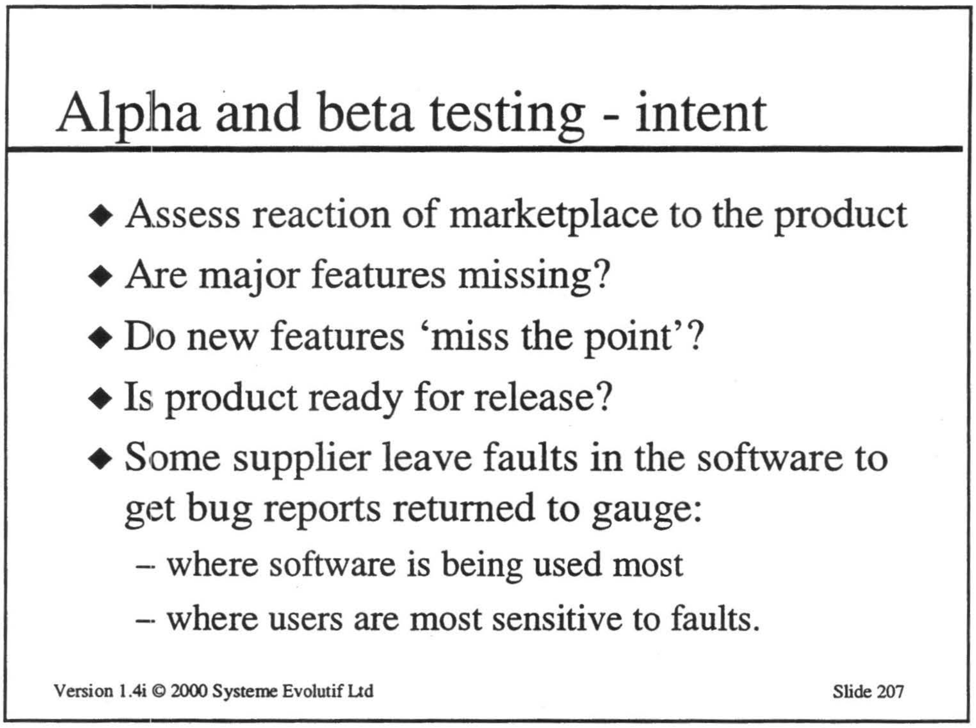
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 205



Alpha and beta testing

- Often used by suppliers of packages (particularly shrink-wrapped)
- Where supplier wishes to receive feedback from actual or potential customers
- Alpha testing normally takes place on the supplier site
- Beta testing usually conducted by users on their site.

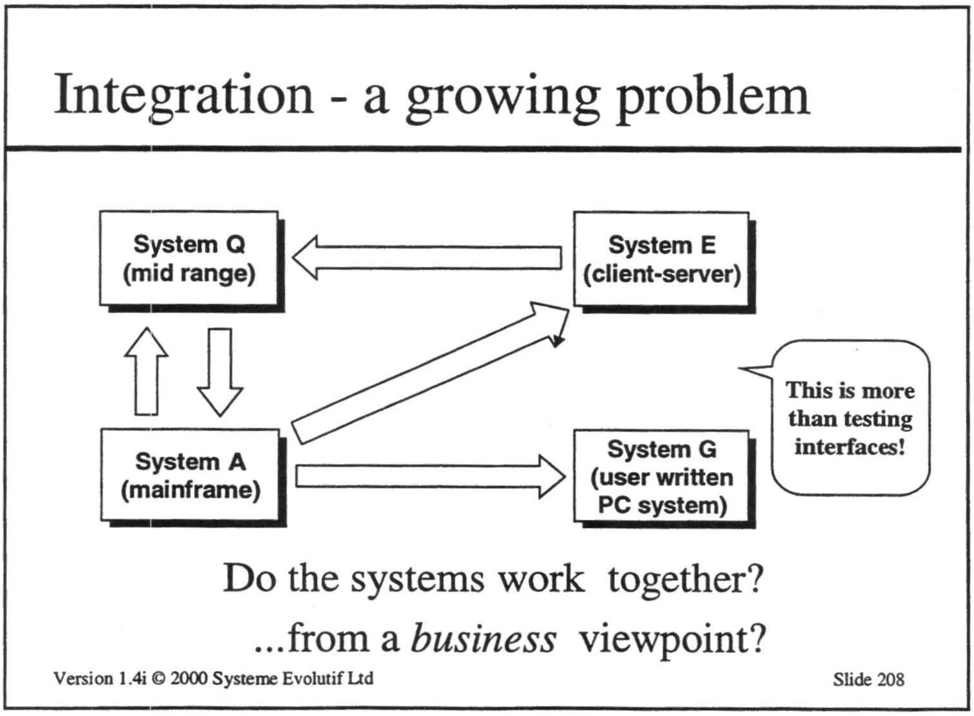
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 206



Alpha and beta testing - intent

- Assess reaction of marketplace to the product
- Are major features missing?
- Do new features 'miss the point'?
- Is product ready for release?
- Some supplier leave faults in the software to get bug reports returned to gauge:
  - where software is being used most
  - where users are most sensitive to faults.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 207



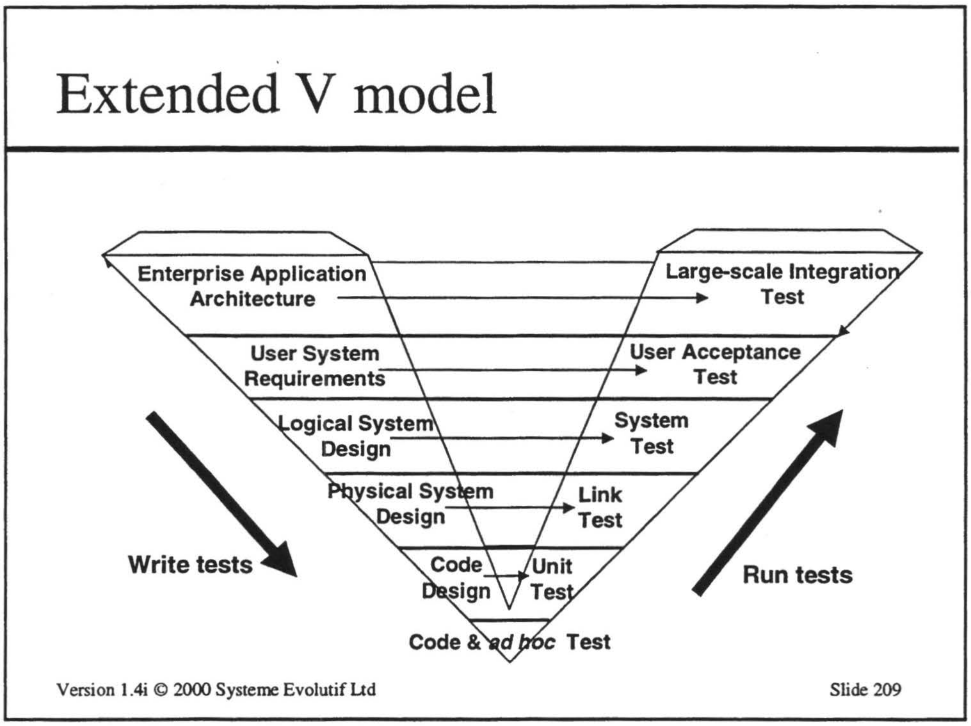
Integration - a growing problem

System Q (mid range)
System E (client-server)
System A (mainframe)
System G (user written PC system)
This is more than testing interfaces!

Do the systems work together?

...from a *business* viewpoint?

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 208



Extended V model

Enterprise Application Architecture → Large-scale Integration Test

User System Requirements → User Acceptance Test

Logical System Design → System Test

Physical System Design → Link Test

Code Design → Unit Test

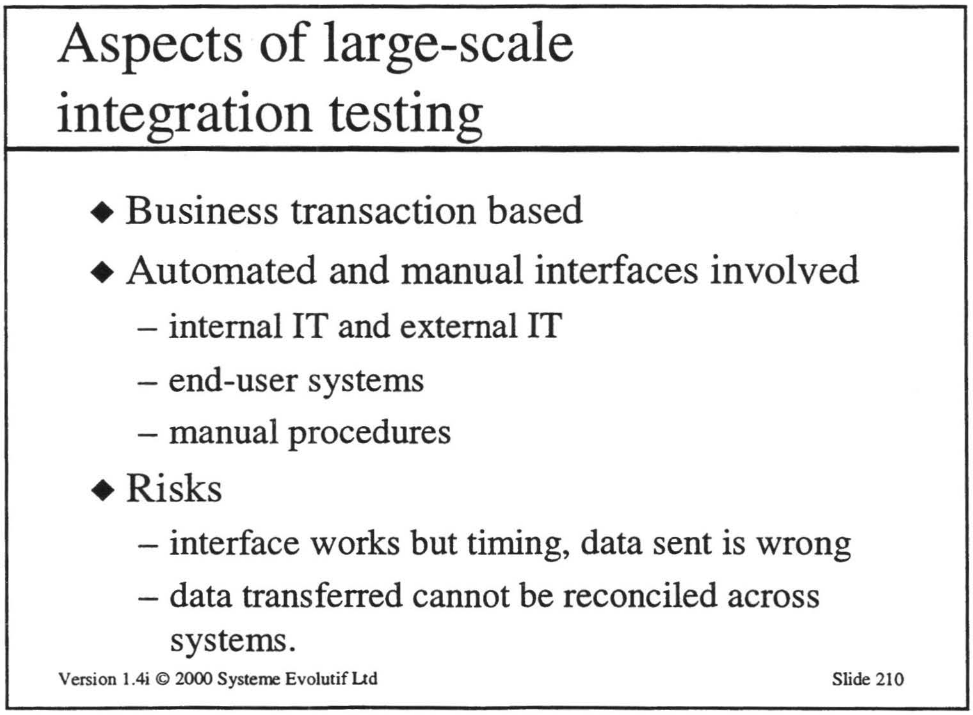
Code & *ad hoc* Test

Write tests

Run tests

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 209

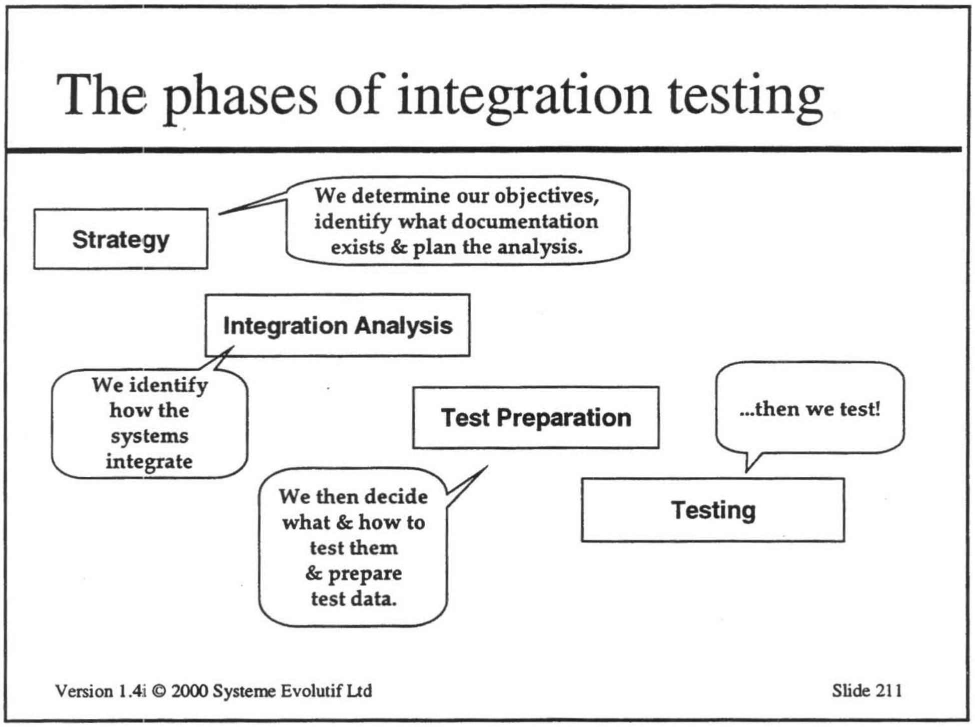


Aspects of large-scale integration testing

- Business transaction based
- Automated and manual interfaces involved
  - internal IT and external IT
  - end-user systems
  - manual procedures
- Risks
  - interface works but timing, data sent is wrong
  - data transferred cannot be reconciled across systems.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 210



The phases of integration testing

Strategy

We determine our objectives, identify what documentation exists & plan the analysis.

Integration Analysis

We identify how the systems integrate

Test Preparation

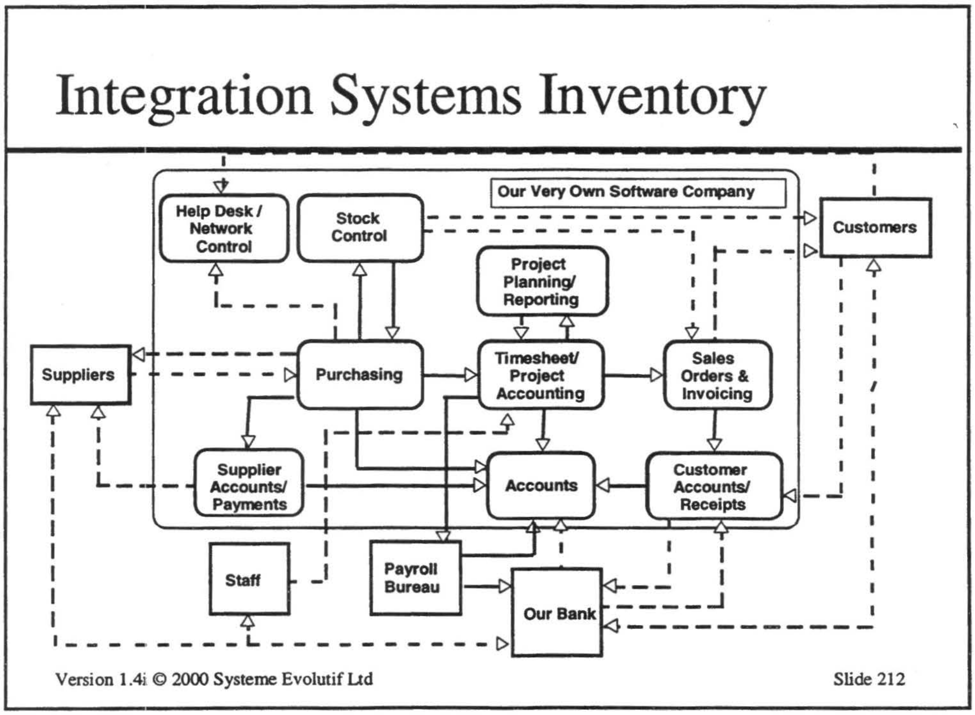
We then decide what & how to test them & prepare test data.

Testing

...then we test!

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 211

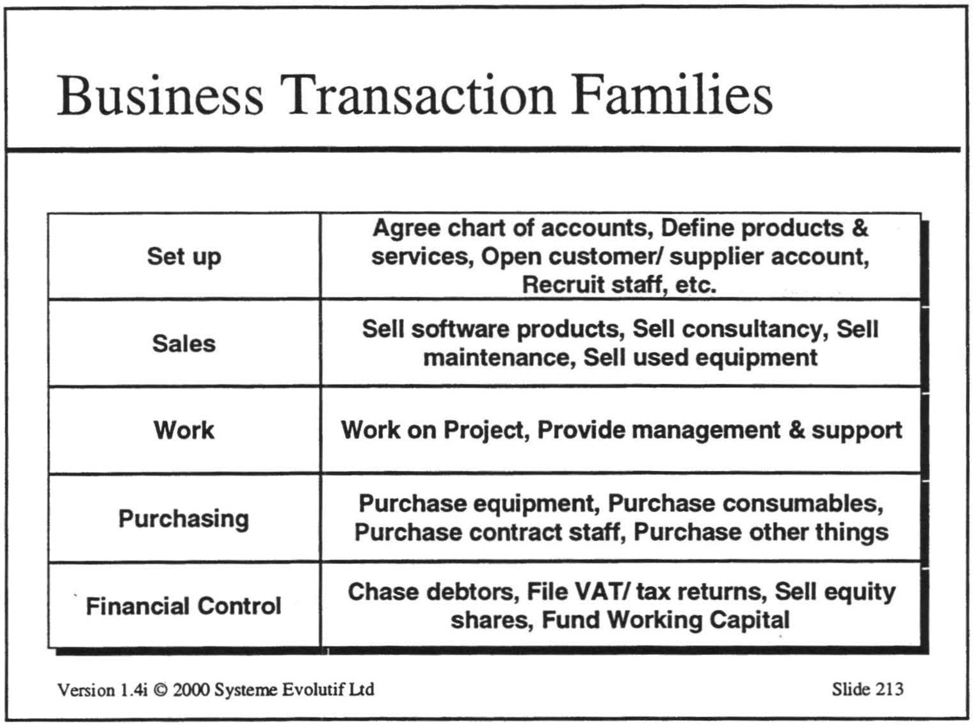


Integration Systems Inventory

Our Very Own Software Company
Help Desk / Network Control
Stock Control
Project Planning/ Reporting
Customers
Suppliers
Purchasing
Timesheet/ Project Accounting
Sales Orders & Invoicing
Supplier Accounts/ Payments
Accounts
Customer Accounts/ Receipts
Staff
Payroll Bureau
Our Bank

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 212

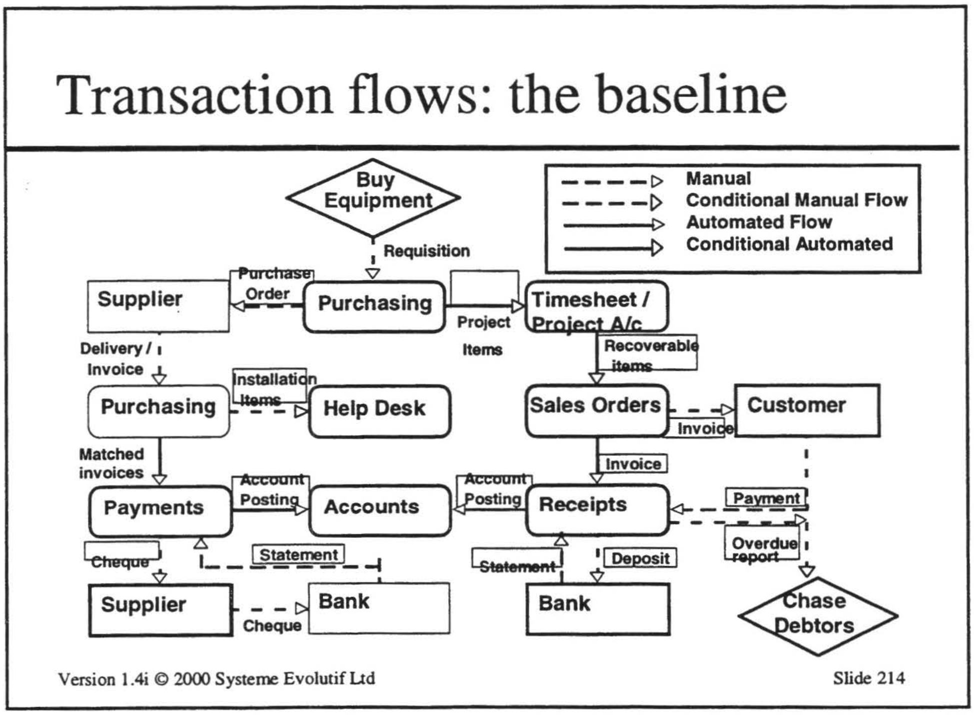


Business Transaction Families

Set up	Agree chart of accounts, Define products & services, Open customer/ supplier account, Recruit staff, etc.
Sales	Sell software products, Sell consultancy, Sell maintenance, Sell used equipment
Work	Work on Project, Provide management & support
Purchasing	Purchase equipment, Purchase consumables, Purchase contract staff, Purchase other things
Financial Control	Chase debtors, File VAT/ tax returns, Sell equity shares, Fund Working Capital

Version 1.4i © 2000 Systeme Evolutif Ltd

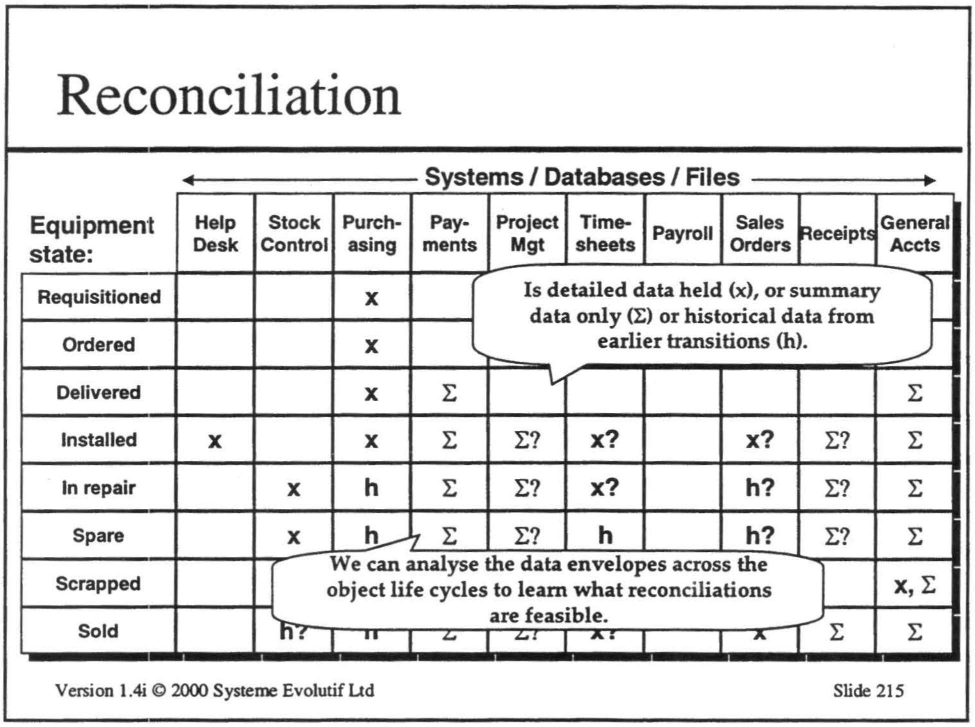
Slide 213



Transaction flows: the baseline

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 214



Reconciliation

Systems / Databases / Files

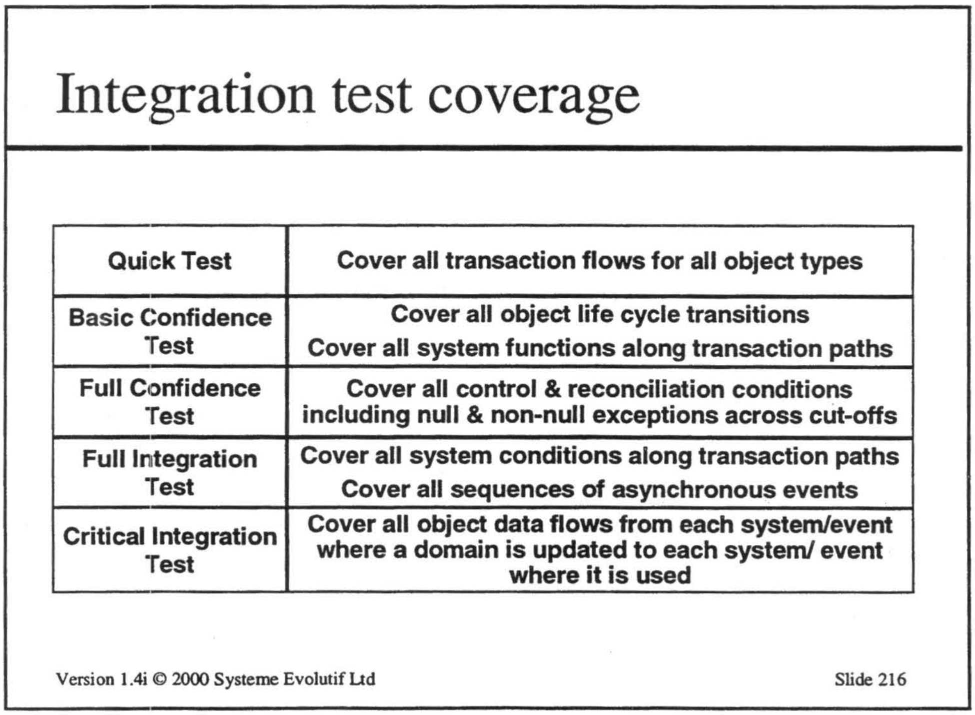
Equipment state:	Help Desk	Stock Control	Purch-asing	Pay-ments	Project Mgt	Time-sheets	Payroll	Sales Orders	Receipts	General Accts
Requisitioned			x							
Ordered			x							
Delivered			x	Σ						Σ
Installed	x		x	Σ	Σ?	x?		x?	Σ?	Σ
In repair		x	h	Σ	Σ?	x?		h?	Σ?	Σ
Spare		x	h	Σ	Σ?	h		h?	Σ?	Σ
Scrapped										x, Σ
Sold		h?	h	Σ	Σ?	x?		x	Σ	Σ

Is detailed data held (x), or summary data only (Σ) or historical data from earlier transitions (h).

We can analyse the data envelopes across the object life cycles to learn what reconciliations are feasible.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 215

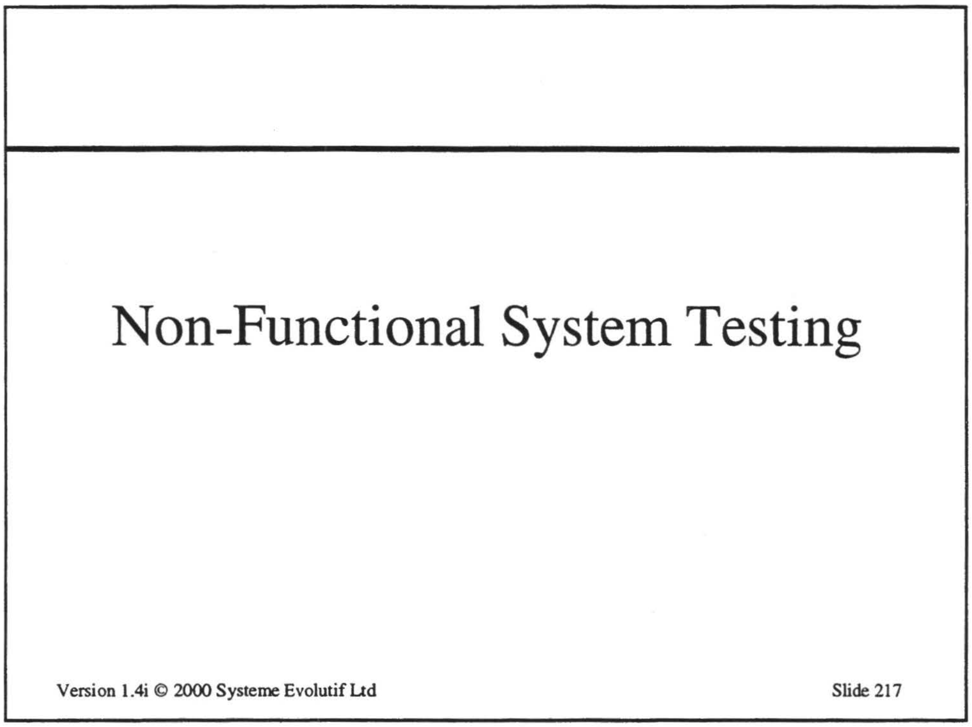


Integration test coverage

Quick Test	Cover all transaction flows for all object types
Basic Confidence Test	Cover all object life cycle transitions Cover all system functions along transaction paths
Full Confidence Test	Cover all control & reconciliation conditions including null & non-null exceptions across cut-offs
Full Integration Test	Cover all system conditions along transaction paths Cover all sequences of asynchronous events
Critical Integration Test	Cover all object data flows from each system/event where a domain is updated to each system/ event where it is used

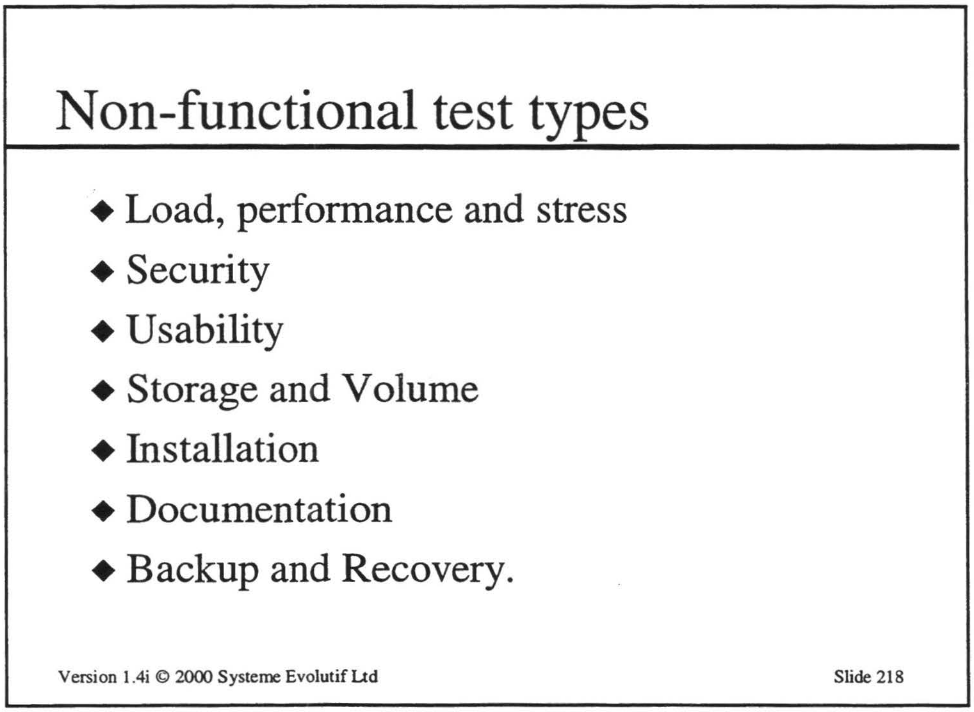
Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 216



Non-Functional System Testing

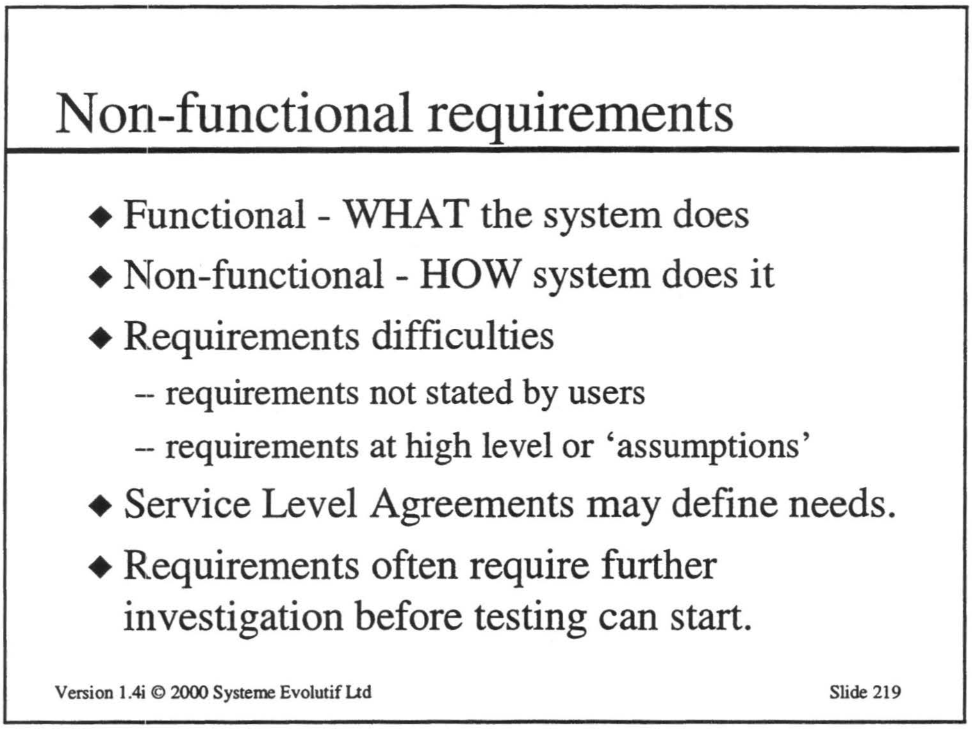
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 217



Non-functional test types

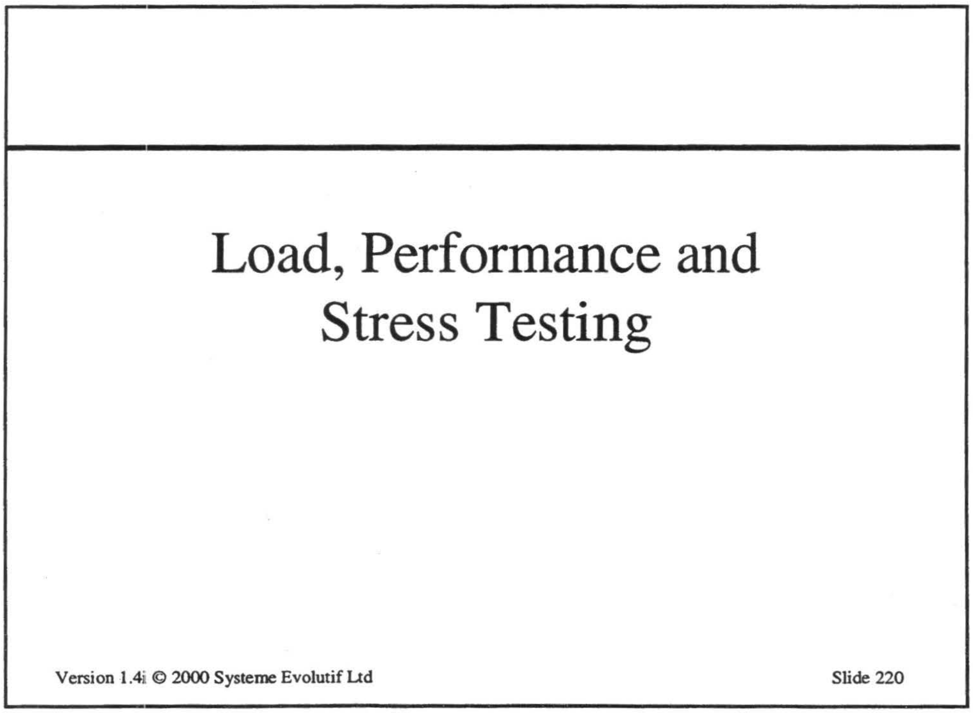
- Load, performance and stress
- Security
- Usability
- Storage and Volume
- Installation
- Documentation
- Backup and Recovery.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 218

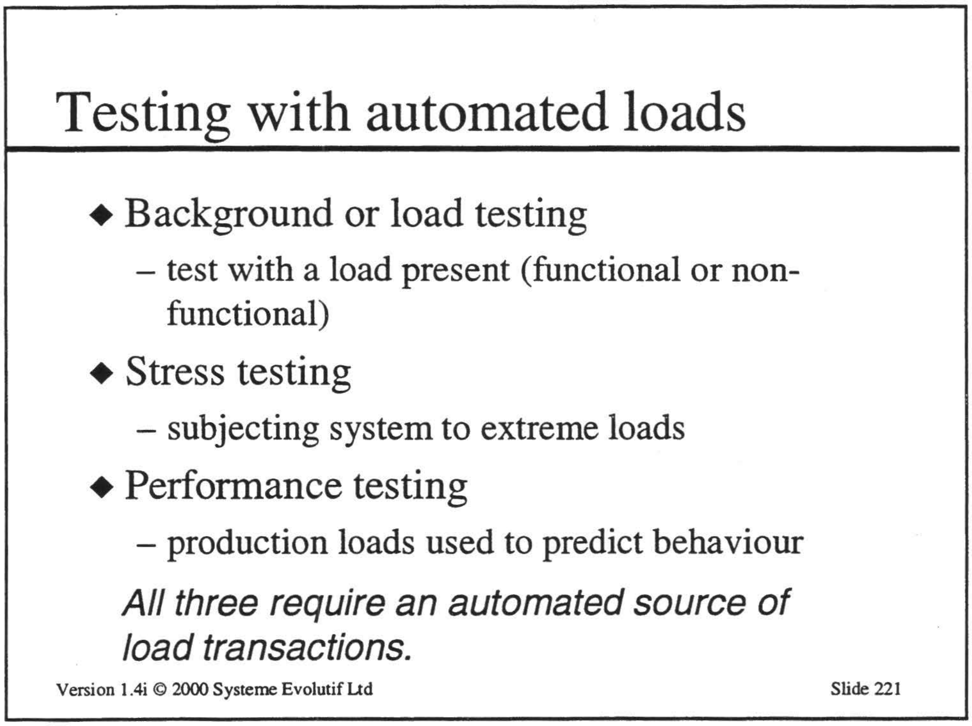


Non-functional requirements

- Functional - WHAT the system does
- Non-functional - HOW system does it
- Requirements difficulties
  - requirements not stated by users
  - requirements at high level or 'assumptions'
- Service Level Agreements may define needs.
- Requirements often require further investigation before testing can start.



Load, Performance and Stress Testing

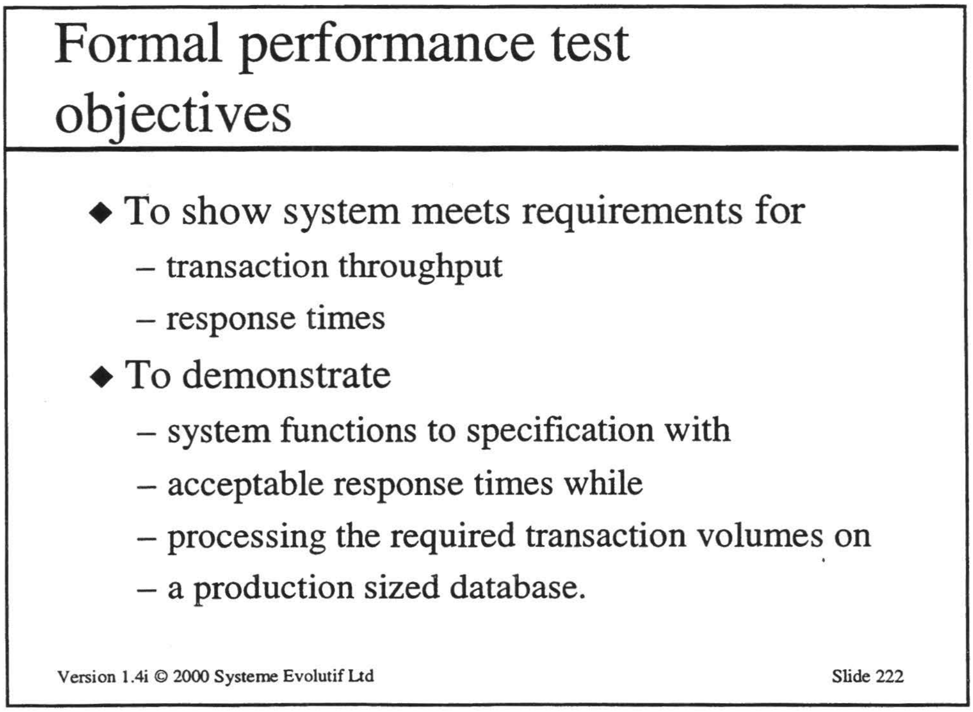


Testing with automated loads

- Background or load testing
  - test with a load present (functional or non-functional)
- Stress testing
  - subjecting system to extreme loads
- Performance testing
  - production loads used to predict behaviour

All three require an automated source of load transactions.

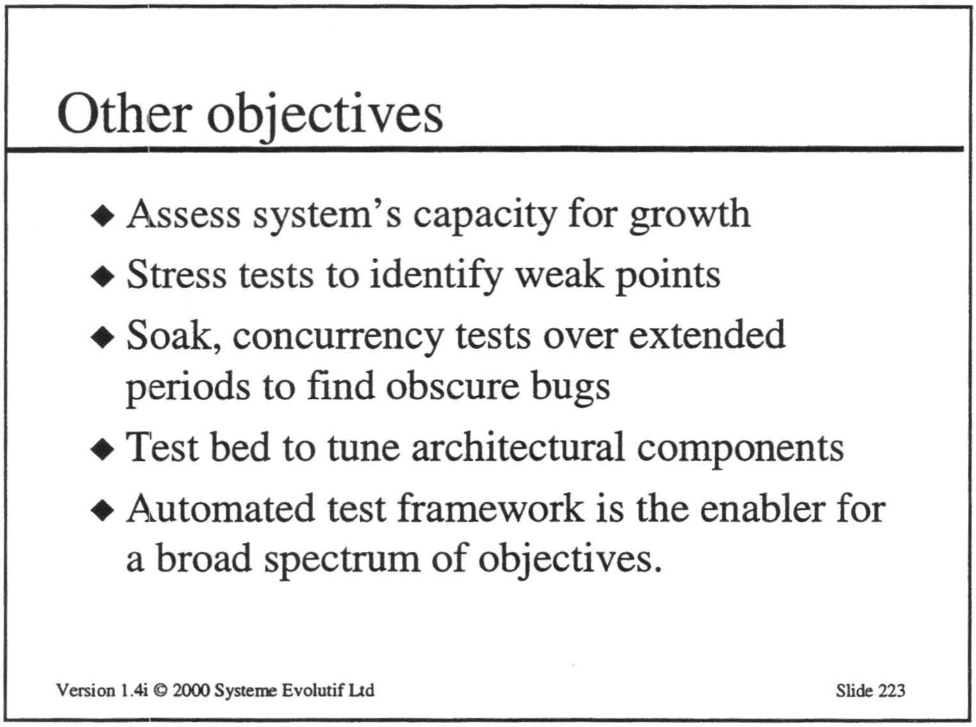
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 221



Formal performance test objectives

- To show system meets requirements for
  - transaction throughput
  - response times
- To demonstrate
  - system functions to specification with
  - acceptable response times while
  - processing the required transaction volumes on
  - a production sized database.

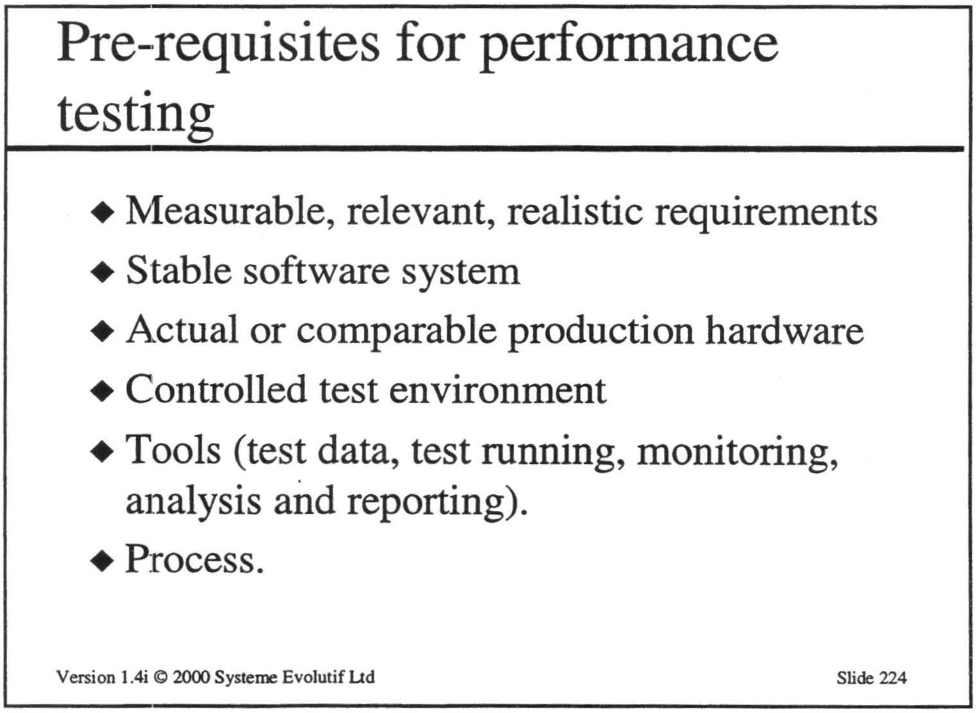
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 222



Other objectives

- ◆ Assess system's capacity for growth
- ◆ Stress tests to identify weak points
- ◆ Soak, concurrency tests over extended periods to find obscure bugs
- ◆ Test bed to tune architectural components
- ◆ Automated test framework is the enabler for a broad spectrum of objectives.

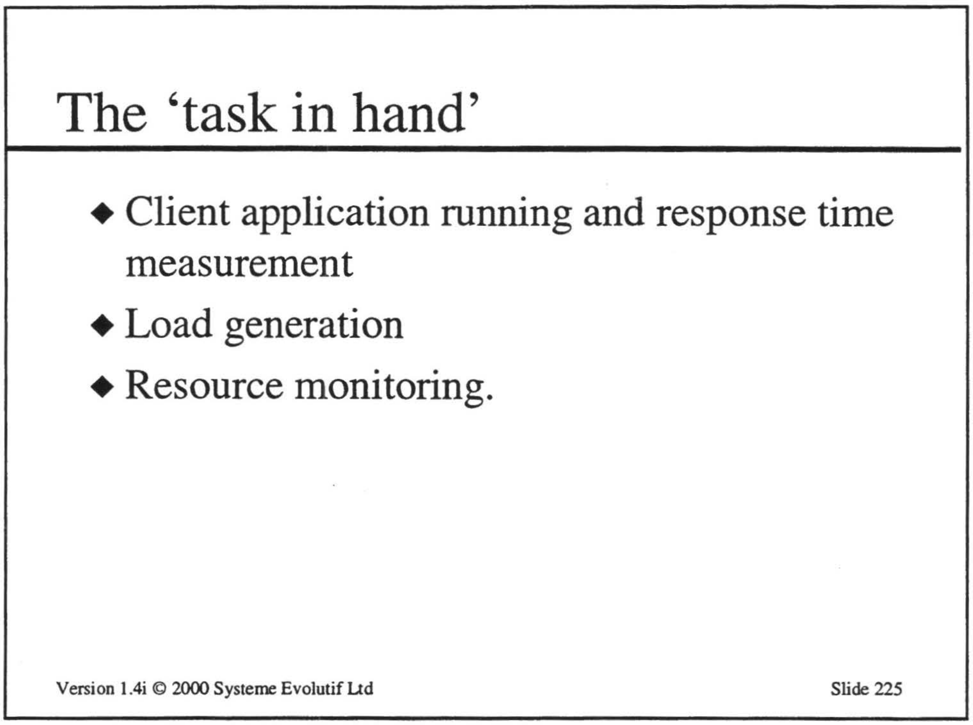
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 223



Pre-requisites for performance testing

- ◆ Measurable, relevant, realistic requirements
- ◆ Stable software system
- ◆ Actual or comparable production hardware
- ◆ Controlled test environment
- ◆ Tools (test data, test running, monitoring, analysis and reporting).
- ◆ Process.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 224

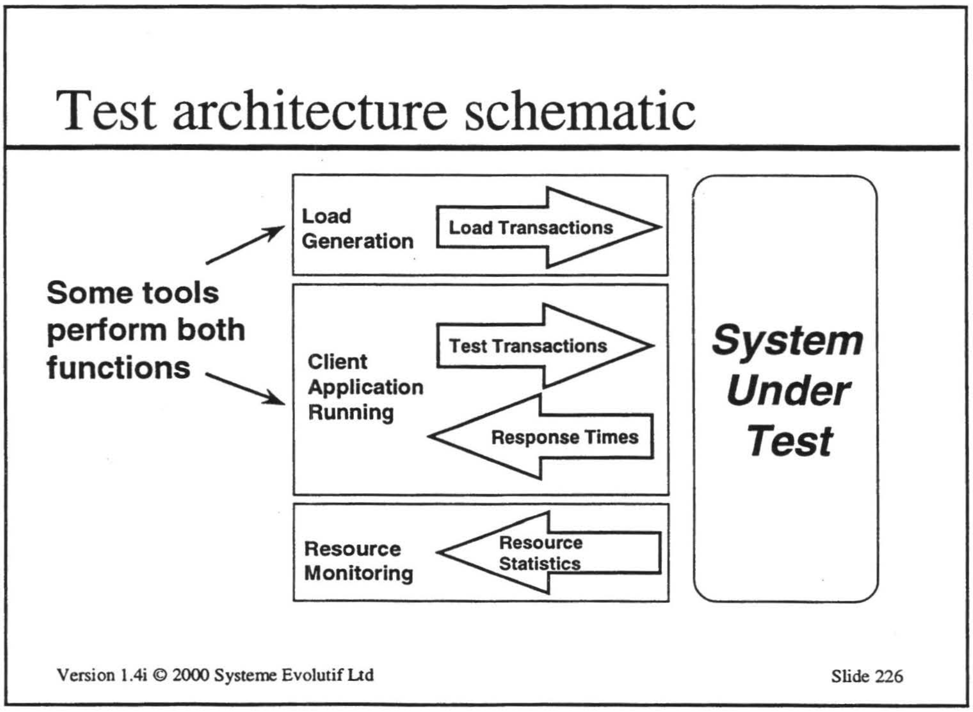


The 'task in hand'

- ◆ Client application running and response time measurement
- ◆ Load generation
- ◆ Resource monitoring.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 225

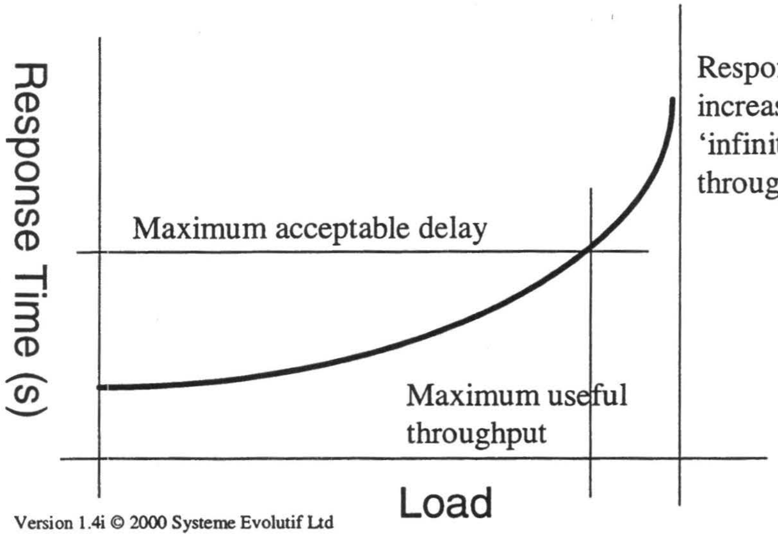


Test architecture schematic

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 226

Response time/load graphs

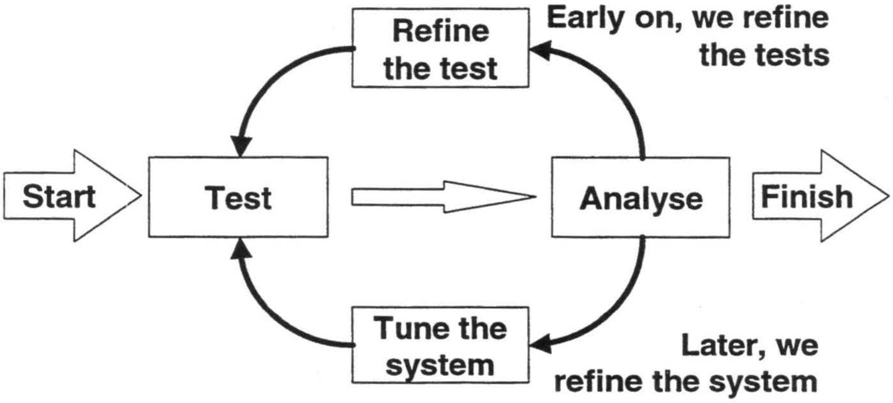


Response times increase to ‘infinity’ at max. throughput

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 227

Test, analyse, refine, tune...cycle

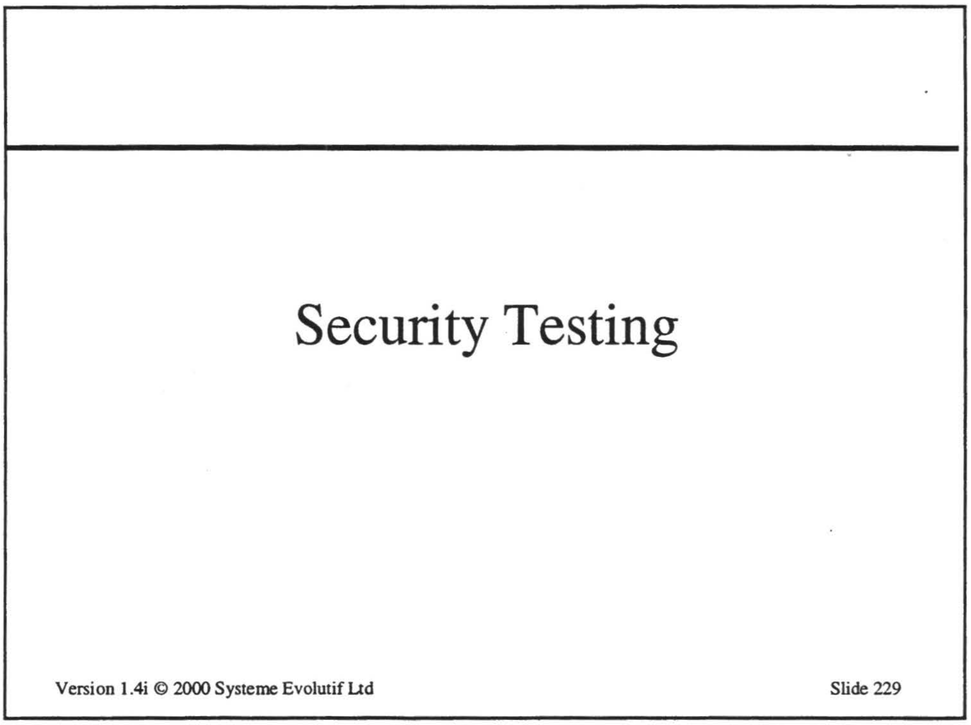


Early on, we refine the tests

Later, we refine the system

Version 1.4i © 2000 Systeme Evolutif Ltd

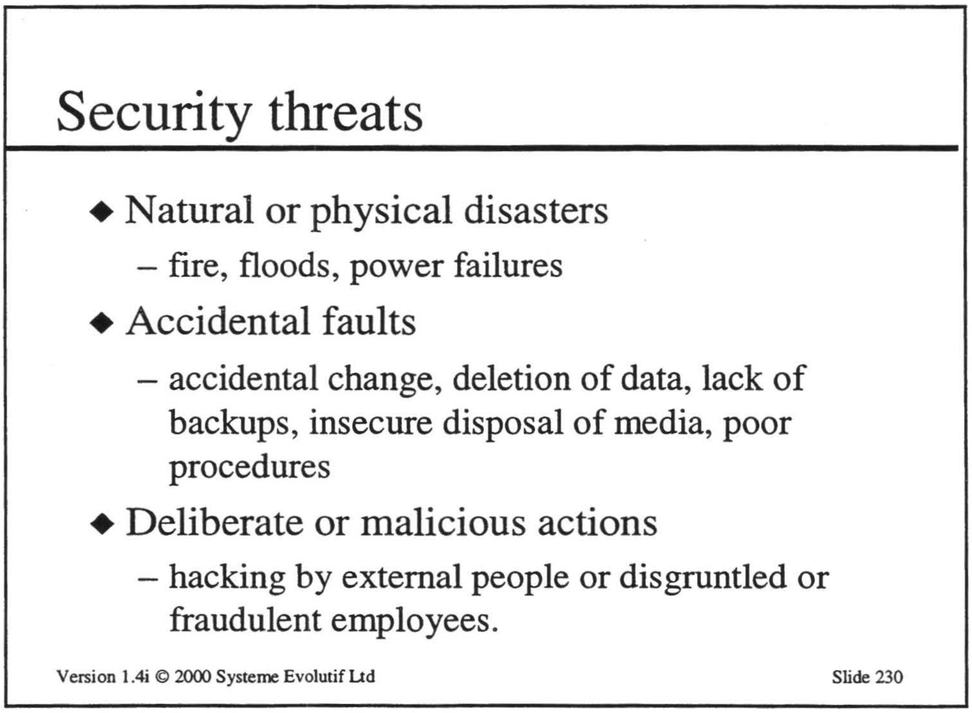
Slide 228



Security Testing

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 229

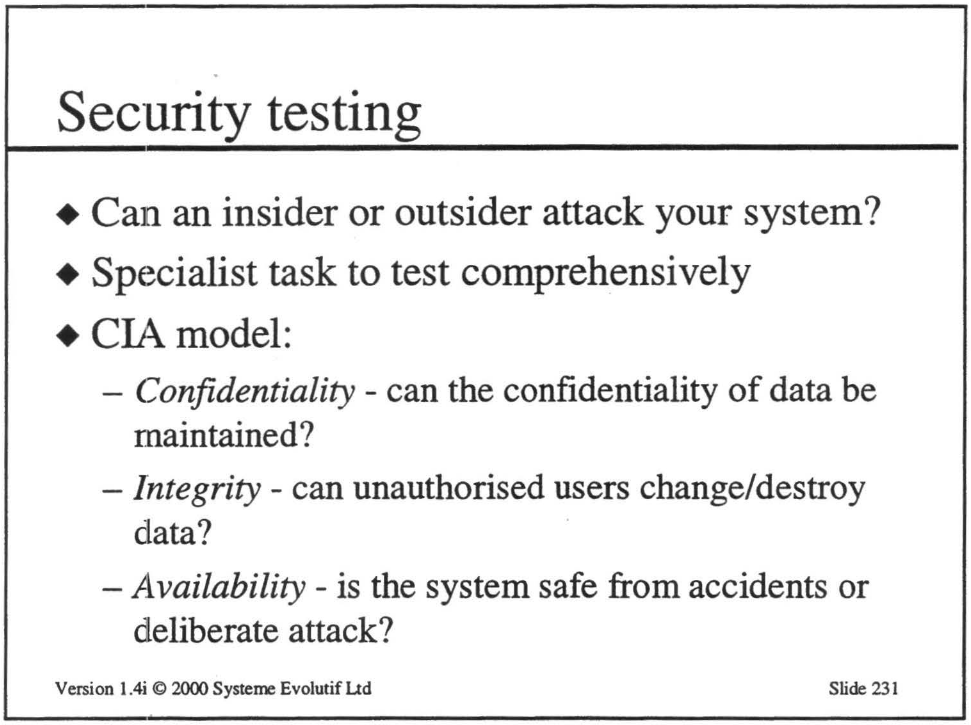


Security threats

- Natural or physical disasters
  - fire, floods, power failures
- Accidental faults
  - accidental change, deletion of data, lack of backups, insecure disposal of media, poor procedures
- Deliberate or malicious actions
  - hacking by external people or disgruntled or fraudulent employees.

Version 1.4i © 2000 Systeme Evolutif Ltd

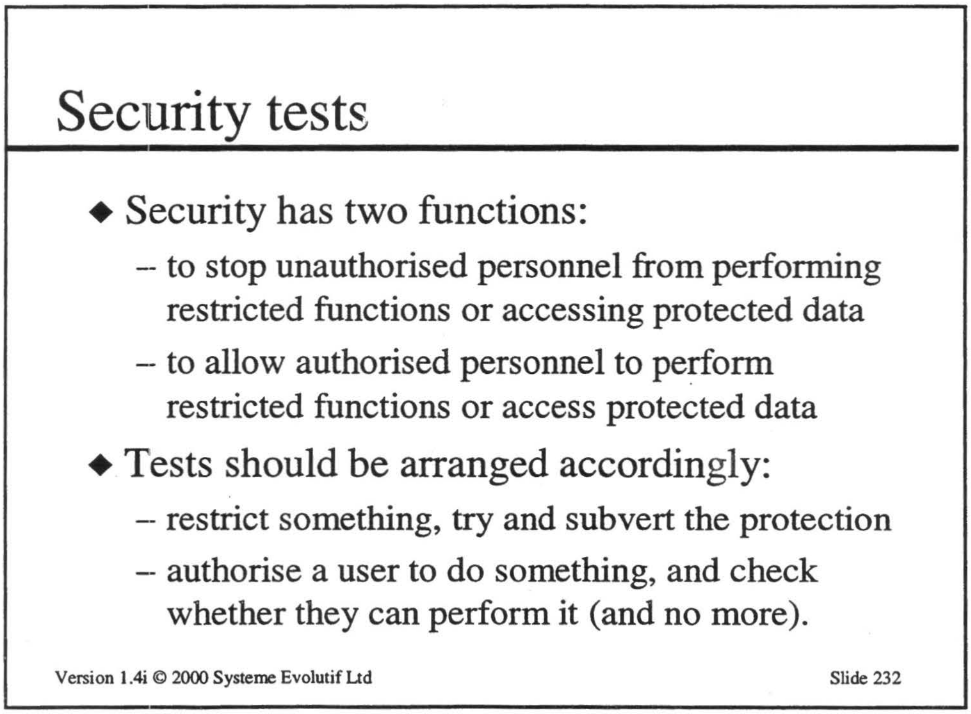
Slide 230



Security testing

- ◆ Can an insider or outsider attack your system?
- ◆ Specialist task to test comprehensively
- ◆ CIA model:
  - – *Confidentiality* - can the confidentiality of data be maintained?
  - – *Integrity* - can unauthorised users change/destroy data?
  - – *Availability* - is the system safe from accidents or deliberate attack?

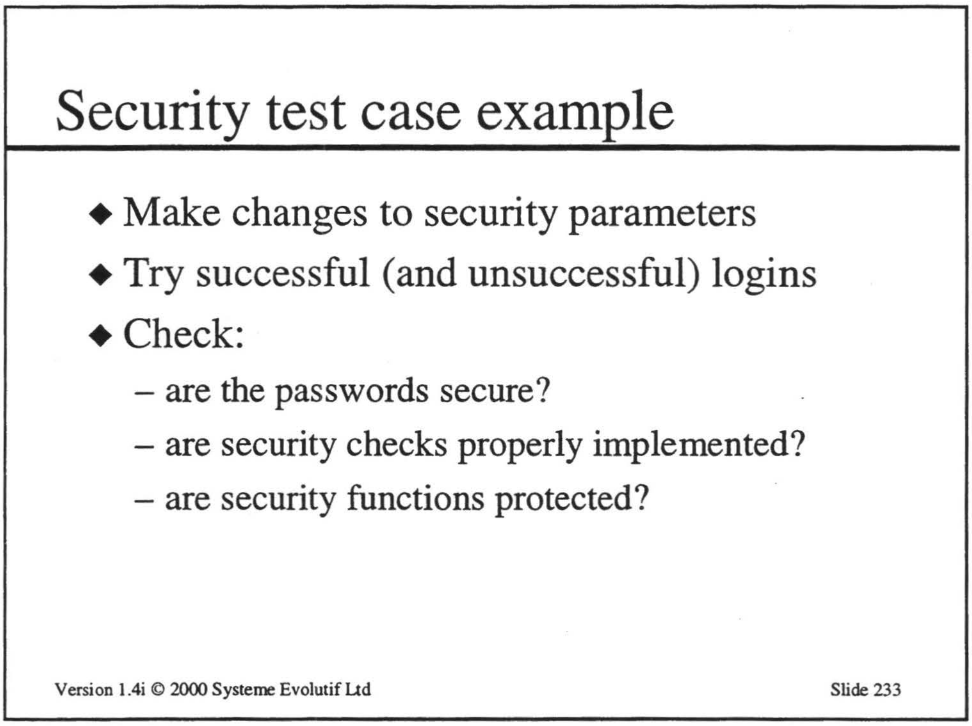
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 231



Security tests

- ◆ Security has two functions:
  - – to stop unauthorised personnel from performing restricted functions or accessing protected data
  - – to allow authorised personnel to perform restricted functions or access protected data
- ◆ Tests should be arranged accordingly:
  - – restrict something, try and subvert the protection
  - – authorise a user to do something, and check whether they can perform it (and no more).

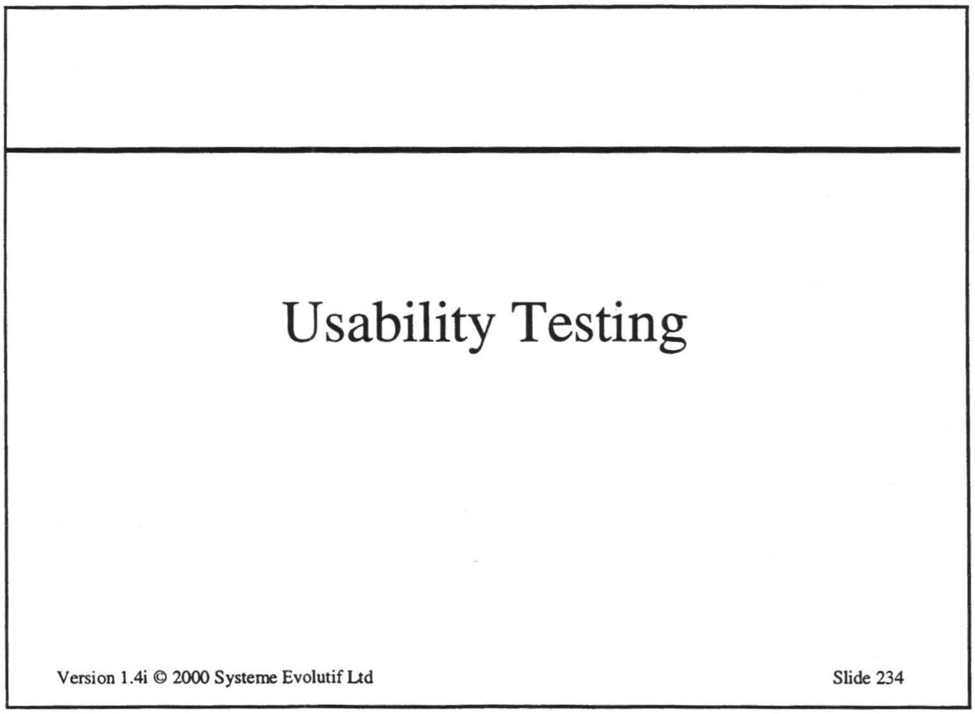
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 232



Security test case example

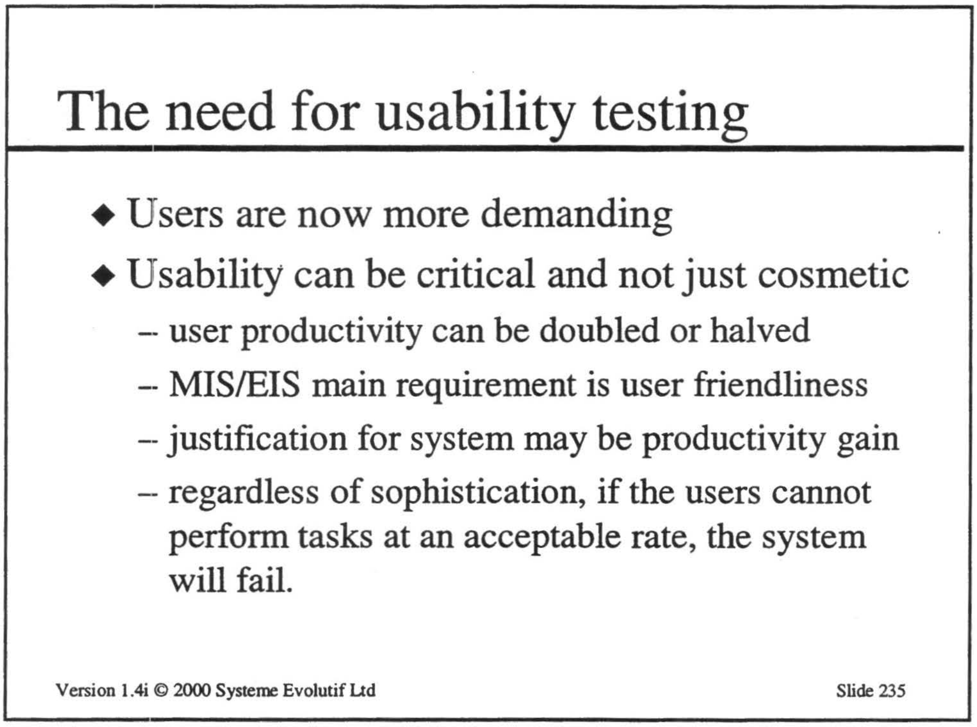
- Make changes to security parameters
- Try successful (and unsuccessful) logins
- Check:
  - are the passwords secure?
  - are security checks properly implemented?
  - are security functions protected?

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 233



Usability Testing

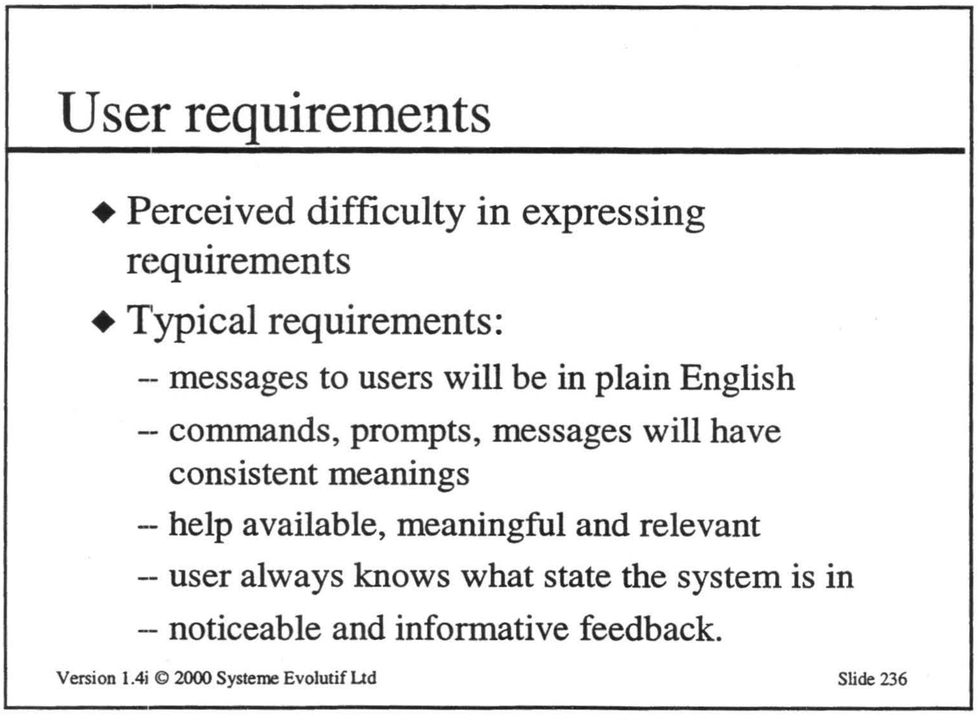
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 234



The need for usability testing

- ◆ Users are now more demanding
- ◆ Usability can be critical and not just cosmetic
  - – user productivity can be doubled or halved
  - – MIS/EIS main requirement is user friendliness
  - – justification for system may be productivity gain
  - – regardless of sophistication, if the users cannot perform tasks at an acceptable rate, the system will fail.

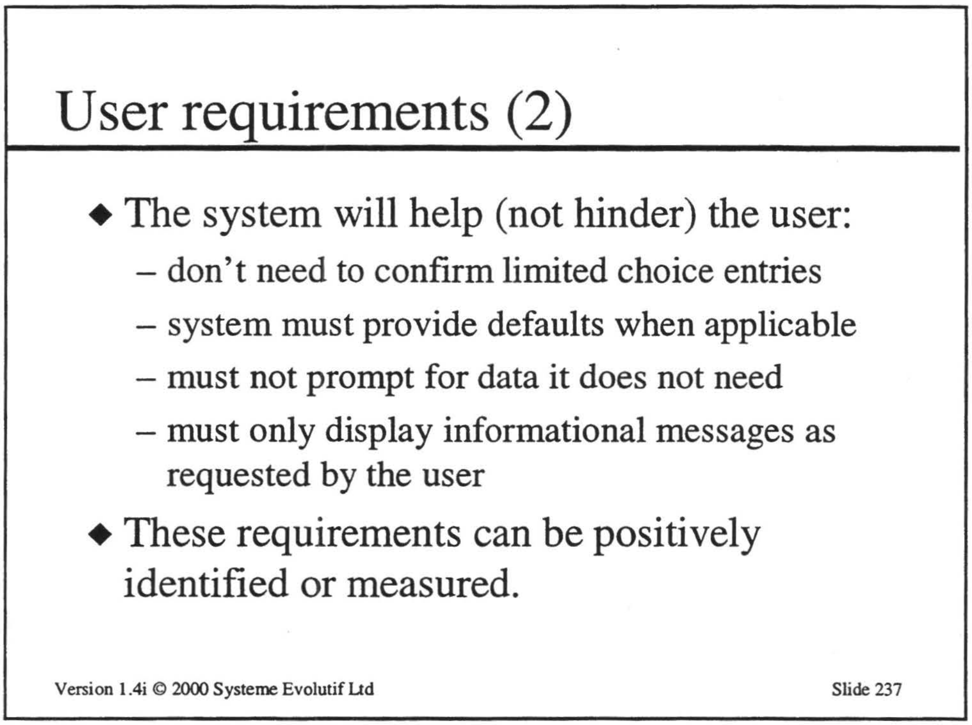
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 235



User requirements

- ◆ Perceived difficulty in expressing requirements
- ◆ Typical requirements:
  - – messages to users will be in plain English
  - – commands, prompts, messages will have consistent meanings
  - – help available, meaningful and relevant
  - – user always knows what state the system is in
  - – noticeable and informative feedback.

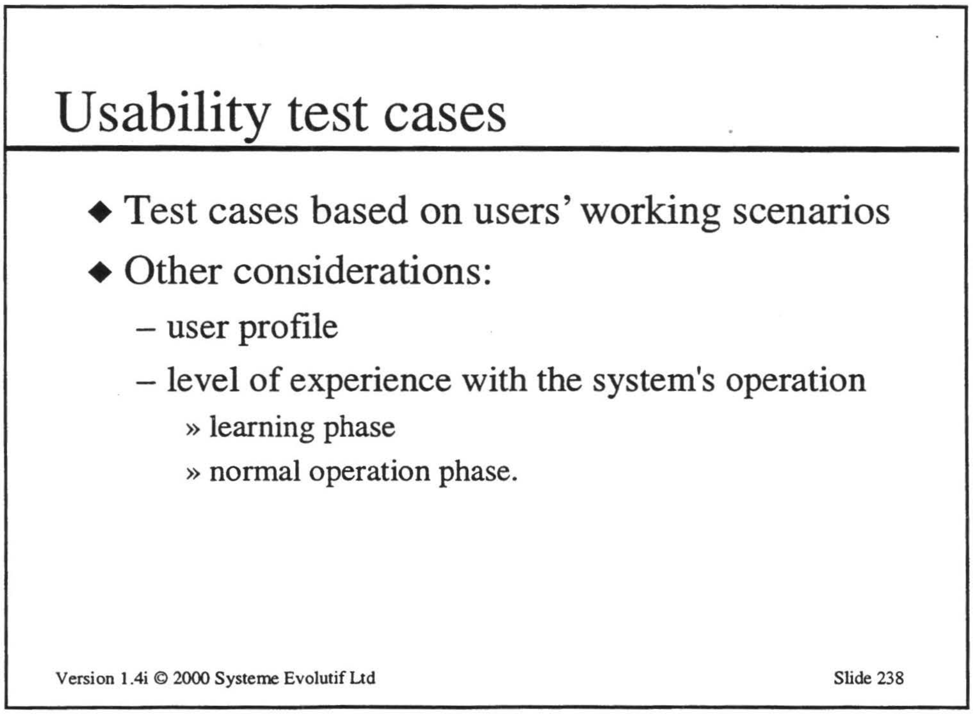
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 236



User requirements (2)

- The system will help (not hinder) the user:
  - don’t need to confirm limited choice entries
  - system must provide defaults when applicable
  - must not prompt for data it does not need
  - must only display informational messages as requested by the user
- These requirements can be positively identified or measured.

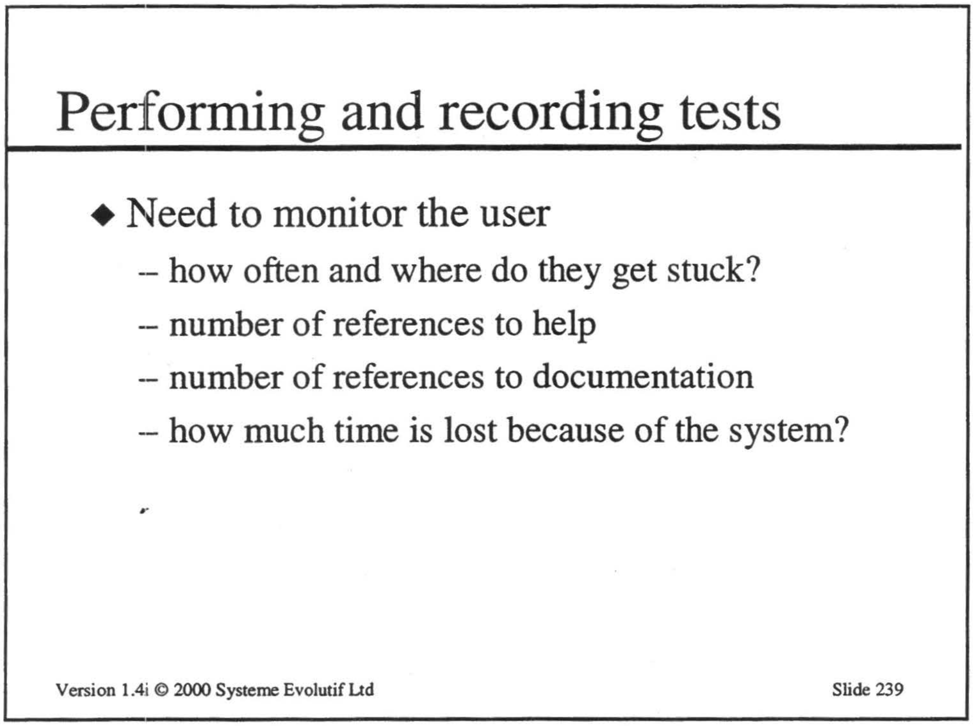
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 237



Usability test cases

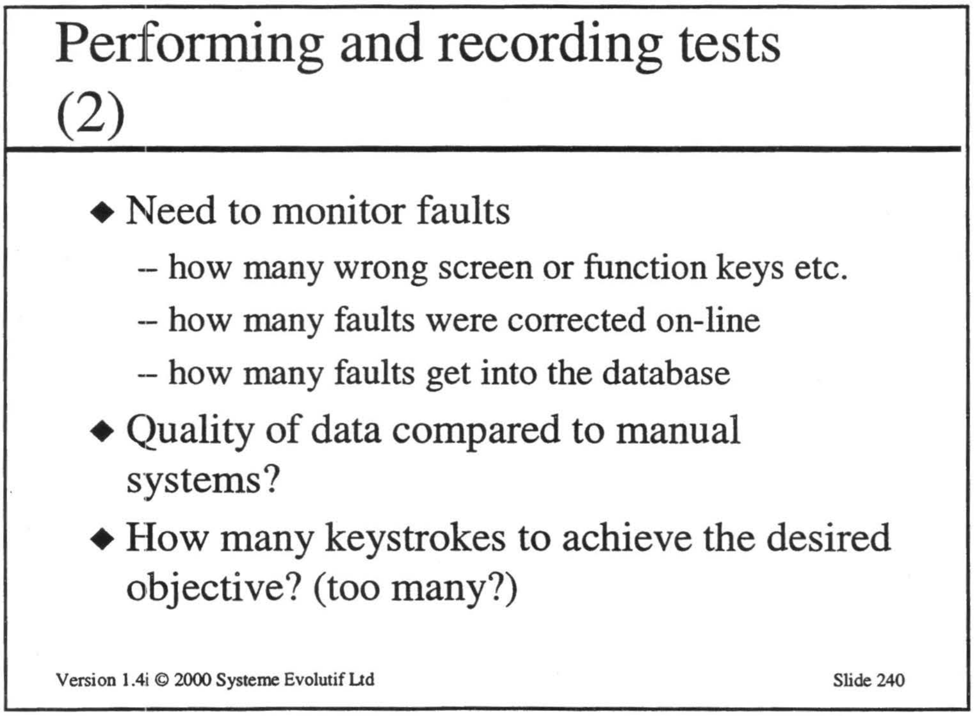
- Test cases based on users’ working scenarios
- Other considerations:
  - user profile
  - level of experience with the system's operation
    - » learning phase
    - » normal operation phase.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 238



Performing and recording tests

- Need to monitor the user
  - how often and where do they get stuck?
  - number of references to help
  - number of references to documentation
  - how much time is lost because of the system?



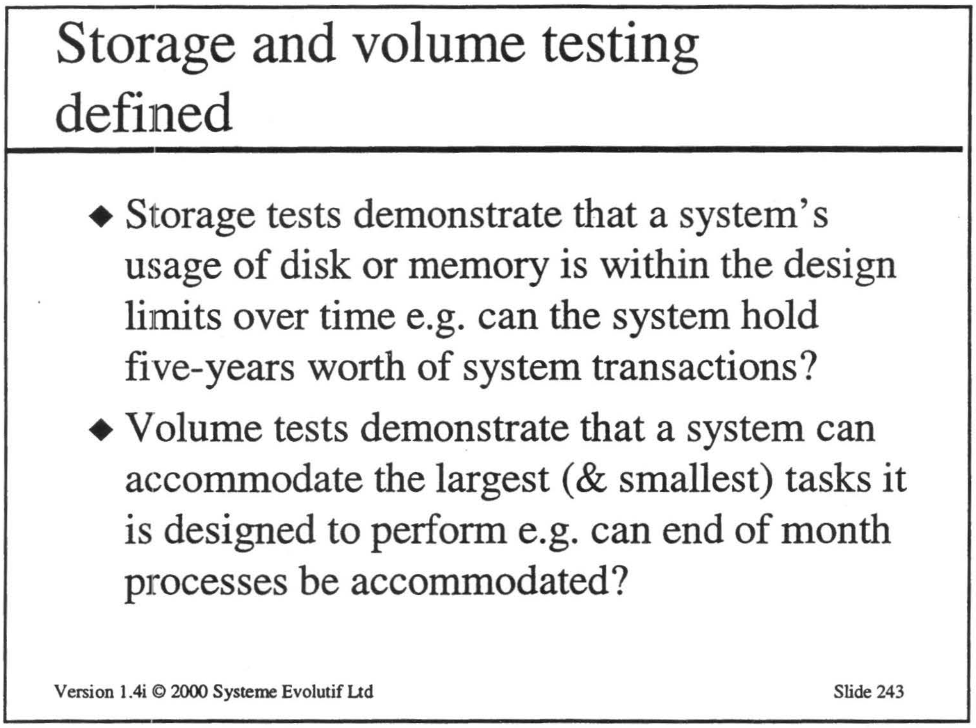
Performing and recording tests (2)

- Need to monitor faults
  - how many wrong screen or function keys etc.
  - how many faults were corrected on-line
  - how many faults get into the database
- Quality of data compared to manual systems?
- How many keystrokes to achieve the desired objective? (too many?)

Satisfaction and frustration factors

- ◆ Users often express frustration - find out why
- ◆ Frustrated expert users
  - – menus or excess detail slows them down?
  - – constant confirmation of trivial entries?
- ◆ Frustrated occasional users
  - – excess options never used?
  - – use help but don't understand, or not relevant
  - – feel they never get feedback, and reassurance.

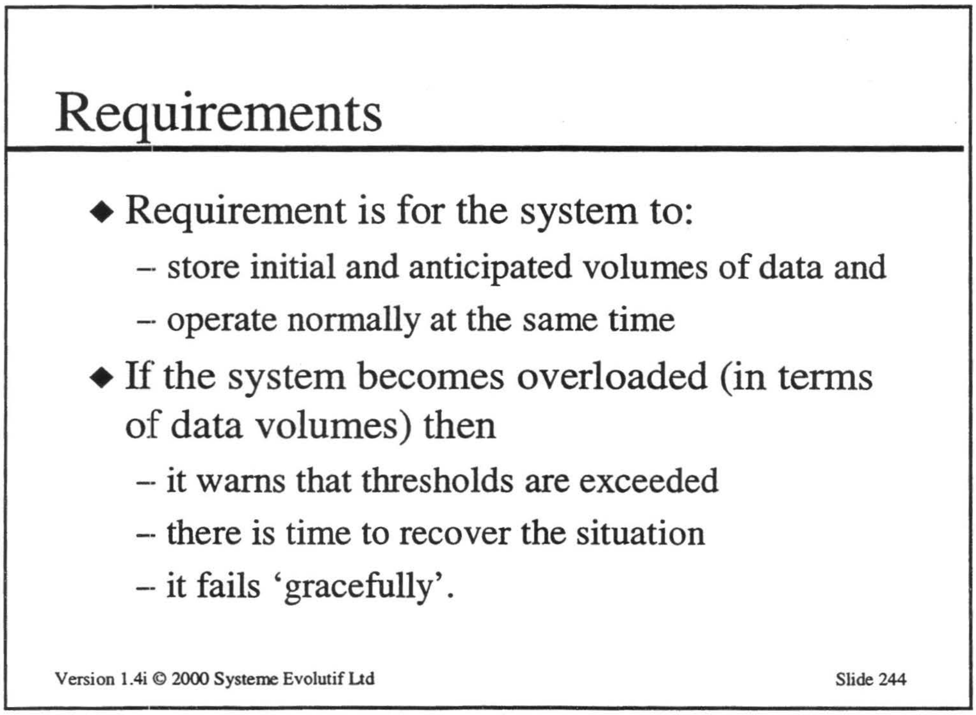
Storage and Volume Testing



Storage and volume testing defined

- Storage tests demonstrate that a system's usage of disk or memory is within the design limits over time e.g. can the system hold five-years worth of system transactions?
- Volume tests demonstrate that a system can accommodate the largest (& smallest) tasks it is designed to perform e.g. can end of month processes be accommodated?

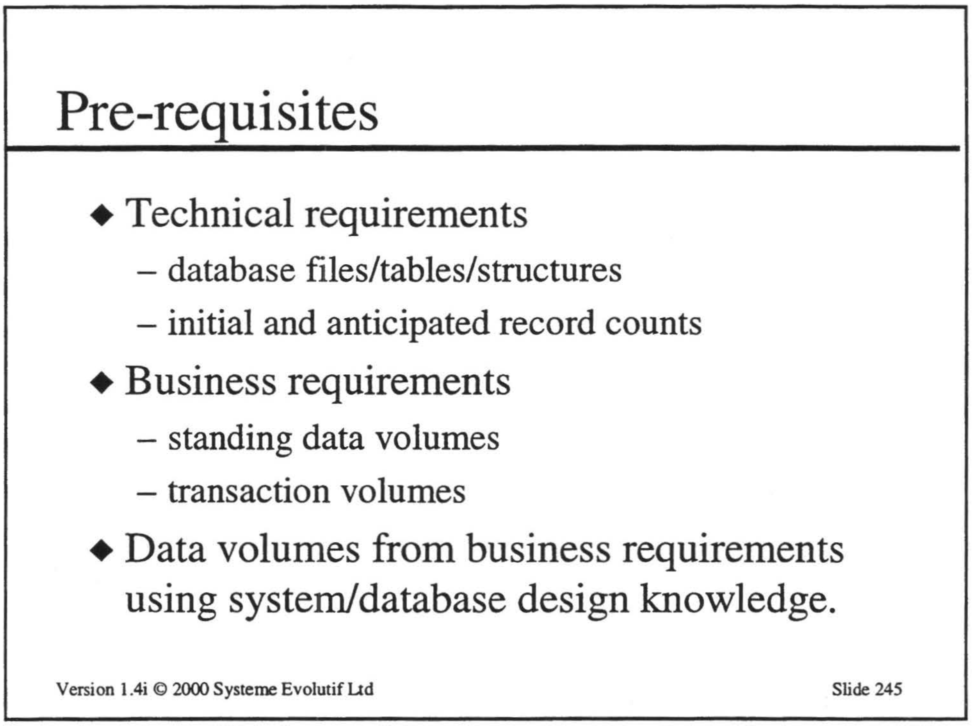
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 243



Requirements

- Requirement is for the system to:
  - store initial and anticipated volumes of data and
  - operate normally at the same time
- If the system becomes overloaded (in terms of data volumes) then
  - it warns that thresholds are exceeded
  - there is time to recover the situation
  - it fails 'gracefully'.

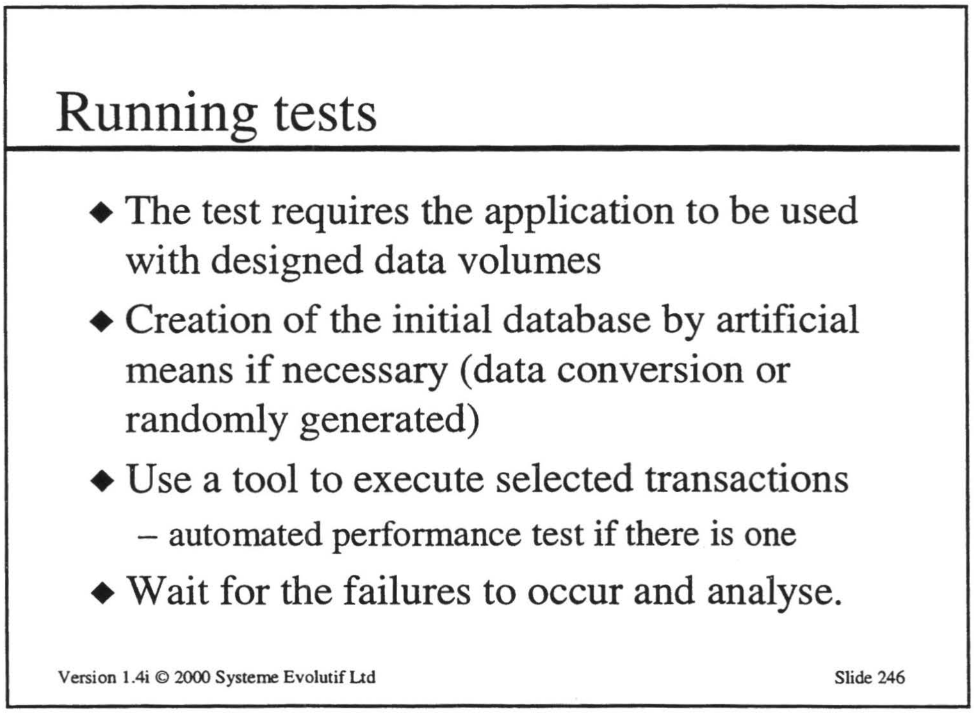
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 244



Pre-requisites

- Technical requirements
  - database files/tables/structures
  - initial and anticipated record counts
- Business requirements
  - standing data volumes
  - transaction volumes
- Data volumes from business requirements using system/database design knowledge.

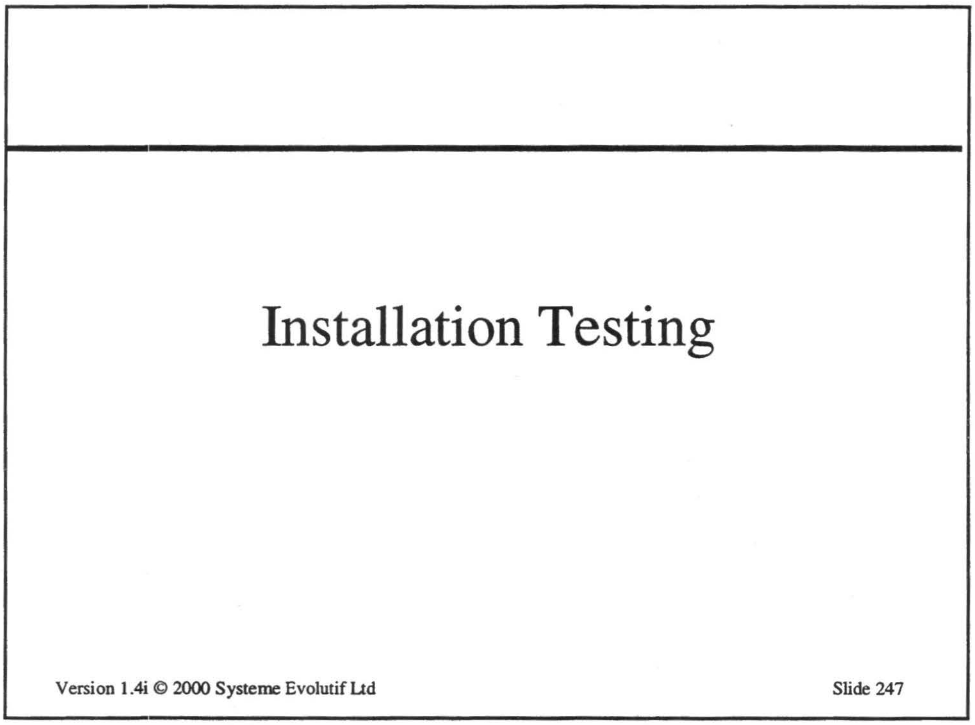
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 245



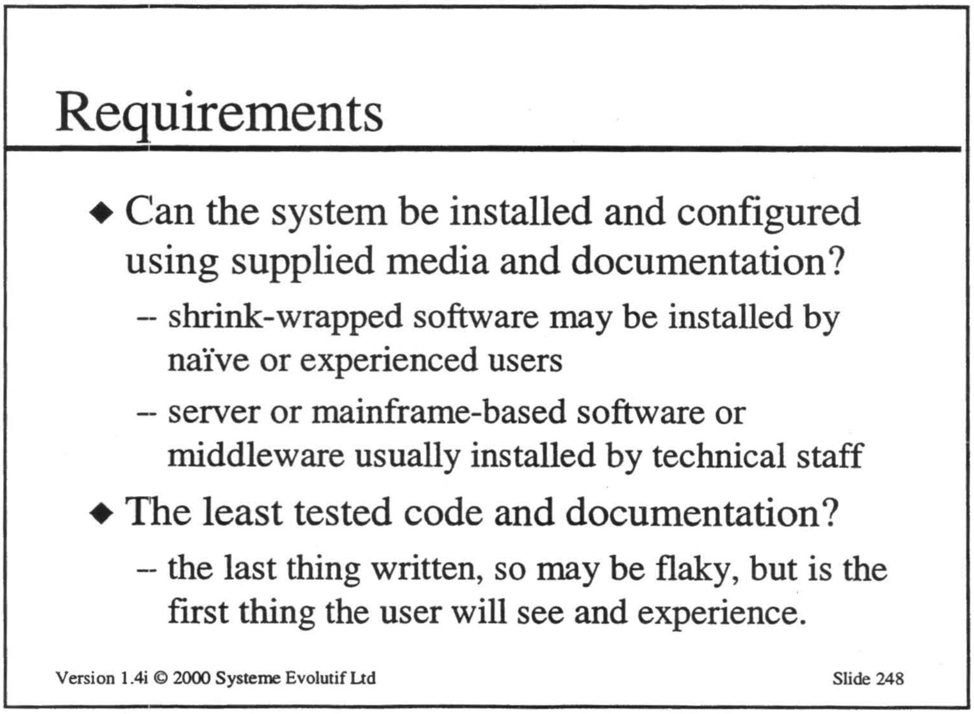
Running tests

- The test requires the application to be used with designed data volumes
- Creation of the initial database by artificial means if necessary (data conversion or randomly generated)
- Use a tool to execute selected transactions
  - automated performance test if there is one
- Wait for the failures to occur and analyse.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 246

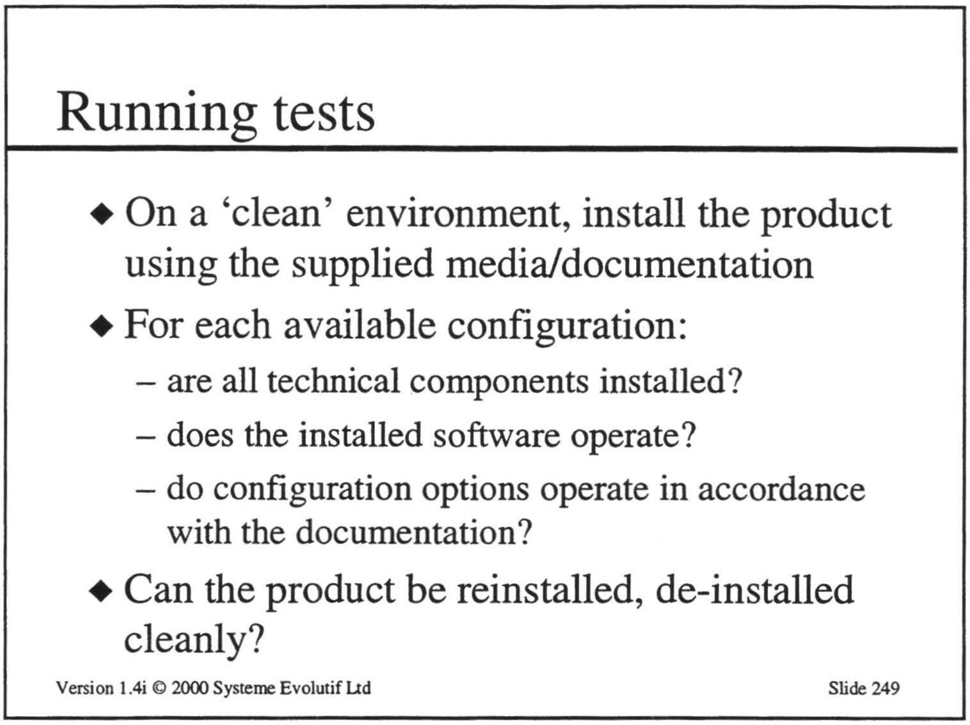


Installation Testing



Requirements

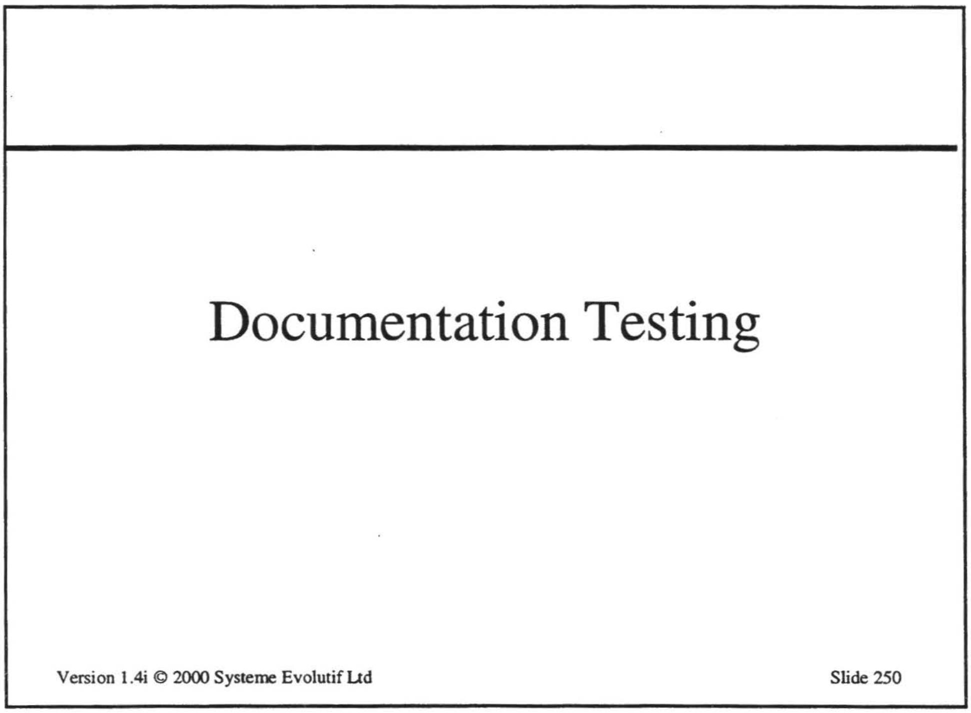
- Can the system be installed and configured using supplied media and documentation?
  - shrink-wrapped software may be installed by naïve or experienced users
  - server or mainframe-based software or middleware usually installed by technical staff
- The least tested code and documentation?
  - the last thing written, so may be flaky, but is the first thing the user will see and experience.



Running tests

- ◆ On a 'clean' environment, install the product using the supplied media/documentation
- ◆ For each available configuration:
  - – are all technical components installed?
  - – does the installed software operate?
  - – do configuration options operate in accordance with the documentation?
- ◆ Can the product be reinstalled, de-installed cleanly?

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 249



Documentation Testing

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 250

Documentation issues

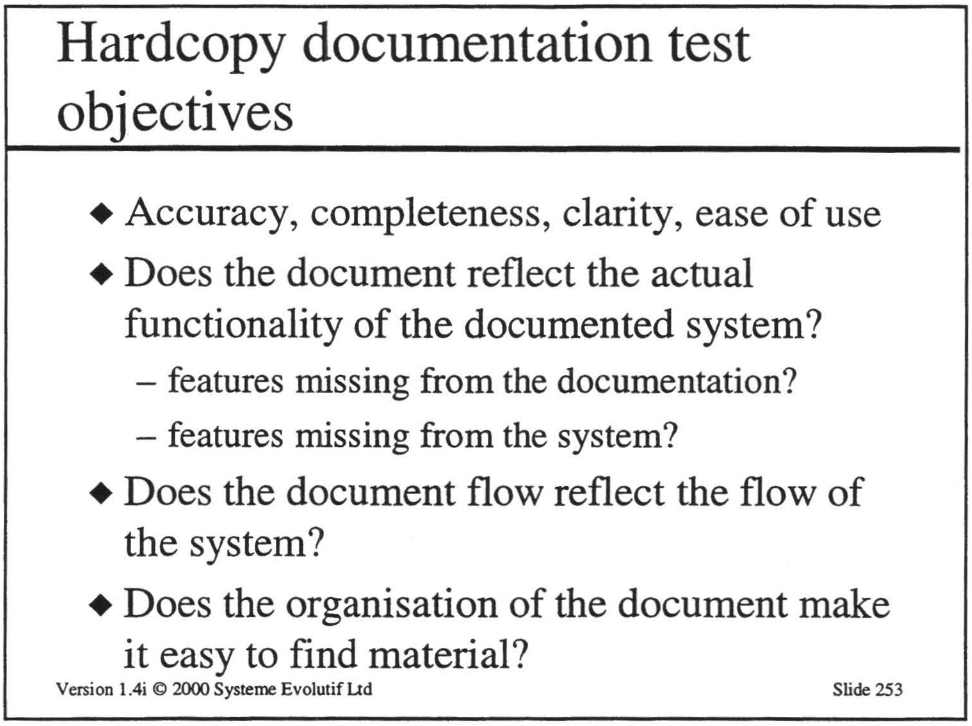
- The product to be tested is more than the software
- Documentation can include:
  - user manuals, quick reference cards
  - installation guides, online help, tutorials, readme files, web site information
  - packaging, sample databases, registration forms, licences, warranty, packing lists…

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 251

Risks of poor documentation

- Software unusable, error prone, slower to use
- Increased costs to the supplier
  - support desk becomes a substitute for the manual
  - many problems turn out to be user errors
  - many ‘enhancements’ requested because the user can’t figure out how to do things
- Bad manuals turn customers off the product
- Users assume software does things it doesn’t and may sue you!

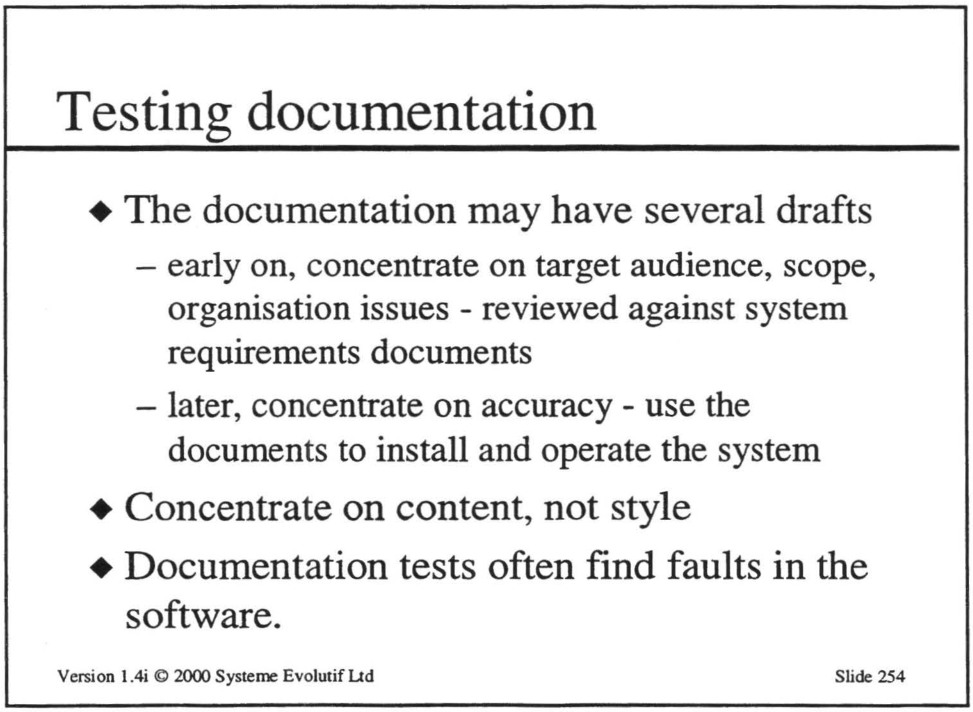
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 252



Hardcopy documentation test objectives

- ◆ Accuracy, completeness, clarity, ease of use
- ◆ Does the document reflect the actual functionality of the documented system?
  - – features missing from the documentation?
  - – features missing from the system?
- ◆ Does the document flow reflect the flow of the system?
- ◆ Does the organisation of the document make it easy to find material?

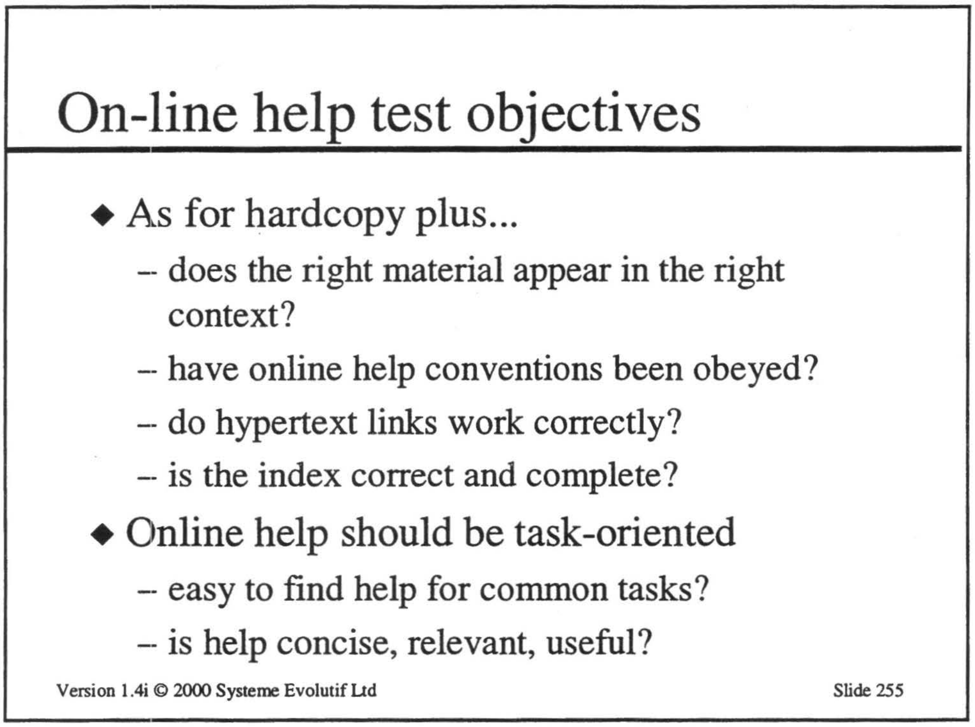
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 253



Testing documentation

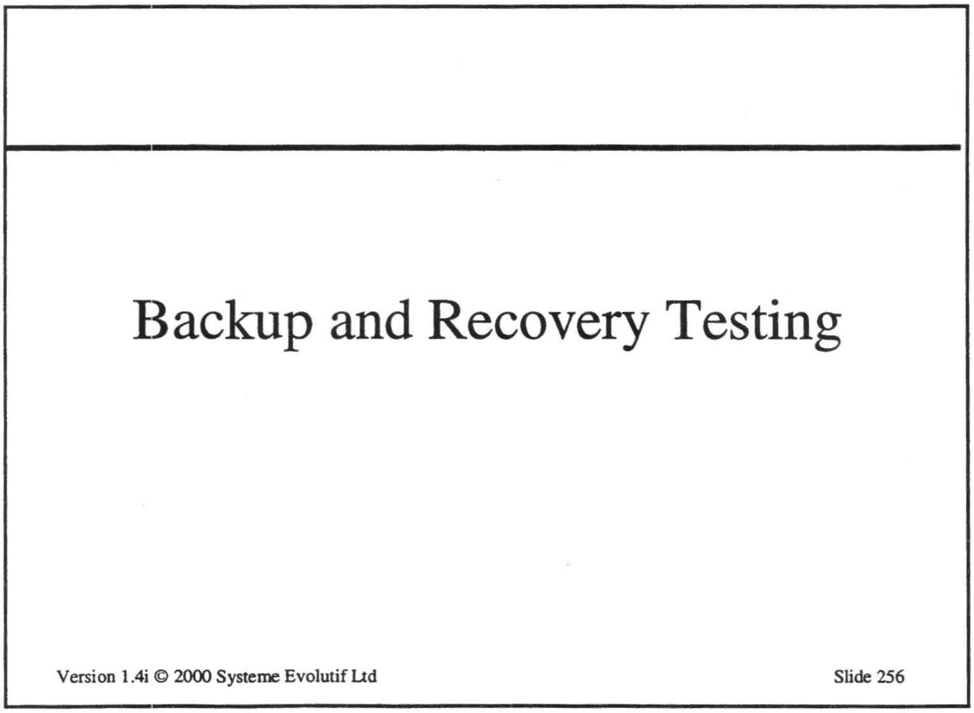
- ◆ The documentation may have several drafts
  - – early on, concentrate on target audience, scope, organisation issues - reviewed against system requirements documents
  - – later, concentrate on accuracy - use the documents to install and operate the system
- ◆ Concentrate on content, not style
- ◆ Documentation tests often find faults in the software.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 254

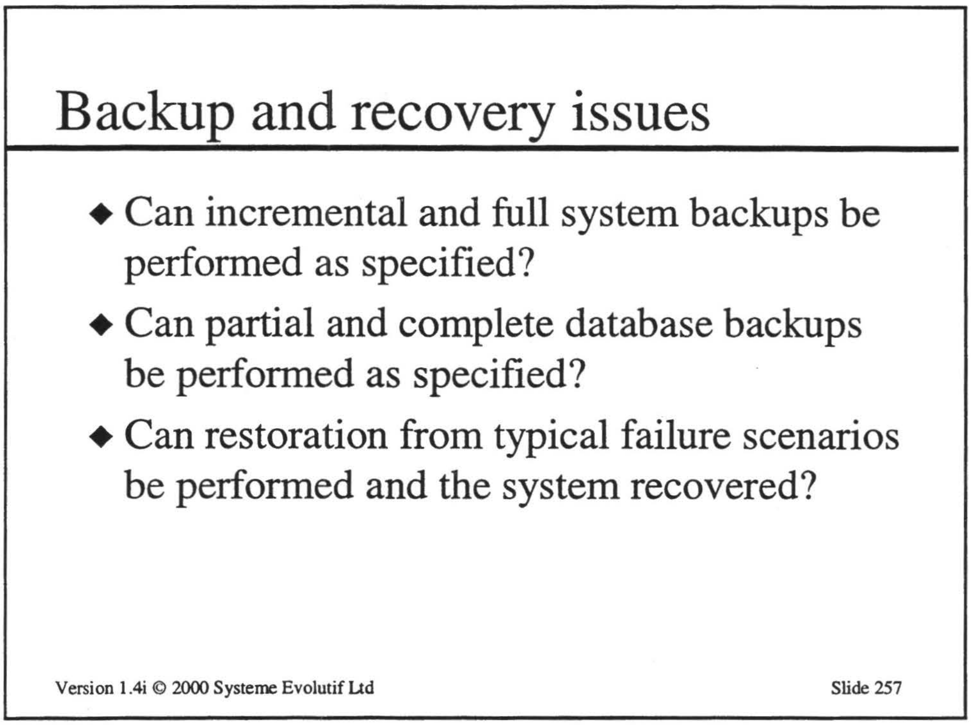


On-line help test objectives

- As for hardcopy plus...
  - does the right material appear in the right context?
  - have online help conventions been obeyed?
  - do hypertext links work correctly?
  - is the index correct and complete?
- Online help should be task-oriented
  - easy to find help for common tasks?
  - is help concise, relevant, useful?



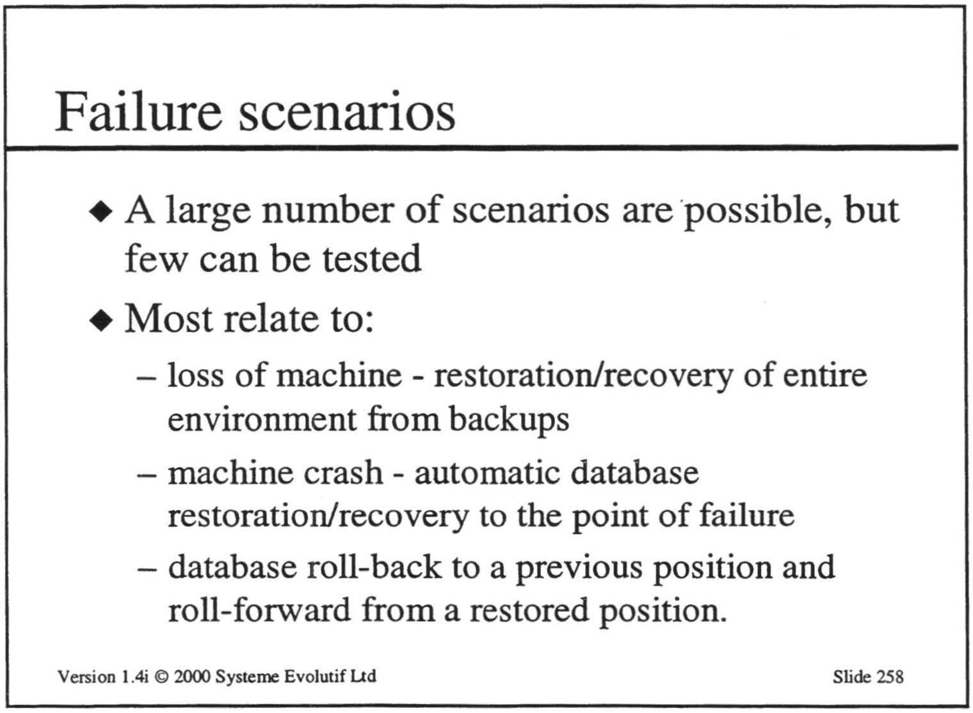
Backup and Recovery Testing



Backup and recovery issues

- Can incremental and full system backups be performed as specified?
- Can partial and complete database backups be performed as specified?
- Can restoration from typical failure scenarios be performed and the system recovered?

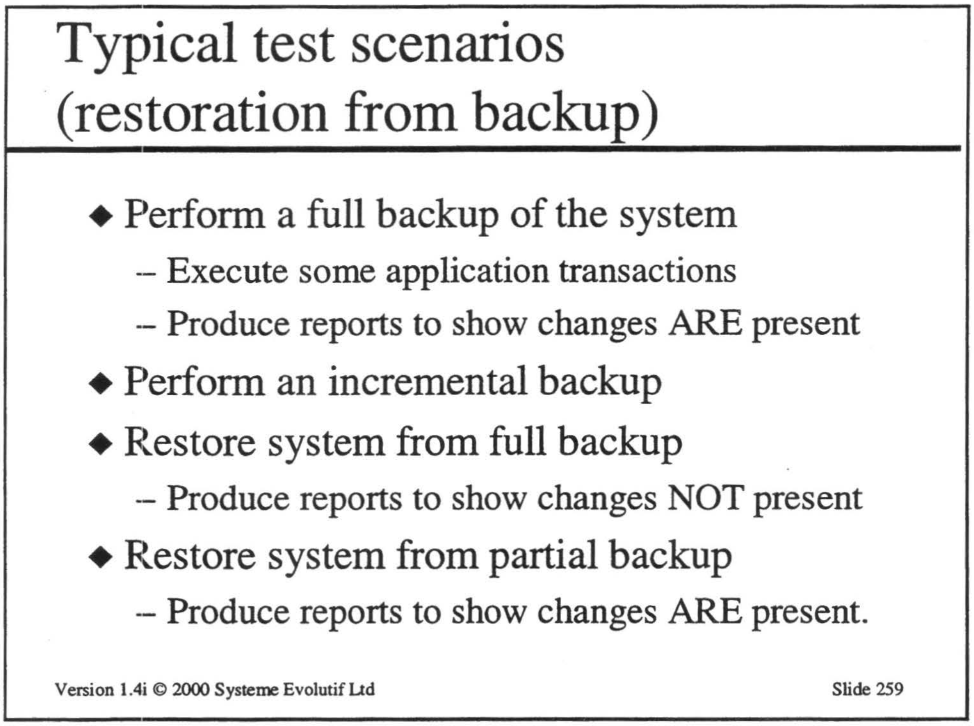
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 257



Failure scenarios

- A large number of scenarios are possible, but few can be tested
- Most relate to:
  - loss of machine - restoration/recovery of entire environment from backups
  - machine crash - automatic database restoration/recovery to the point of failure
  - database roll-back to a previous position and roll-forward from a restored position.

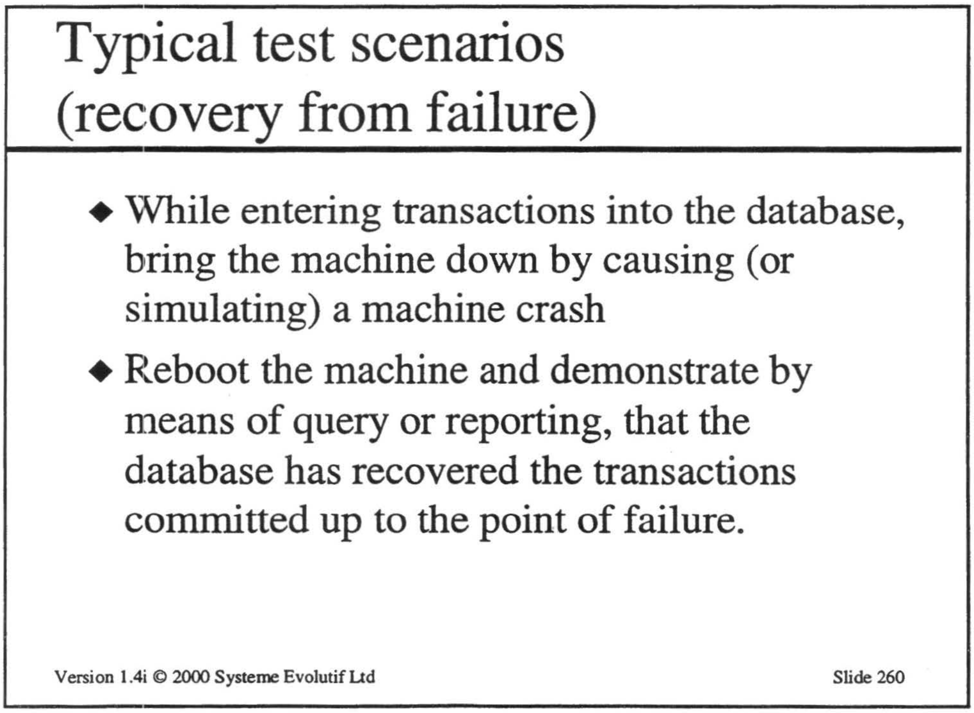
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 258



Typical test scenarios (restoration from backup)

- ◆ Perform a full backup of the system
  - – Execute some application transactions
  - – Produce reports to show changes ARE present
- ◆ Perform an incremental backup
- ◆ Restore system from full backup
  - – Produce reports to show changes NOT present
- ◆ Restore system from partial backup
  - – Produce reports to show changes ARE present.

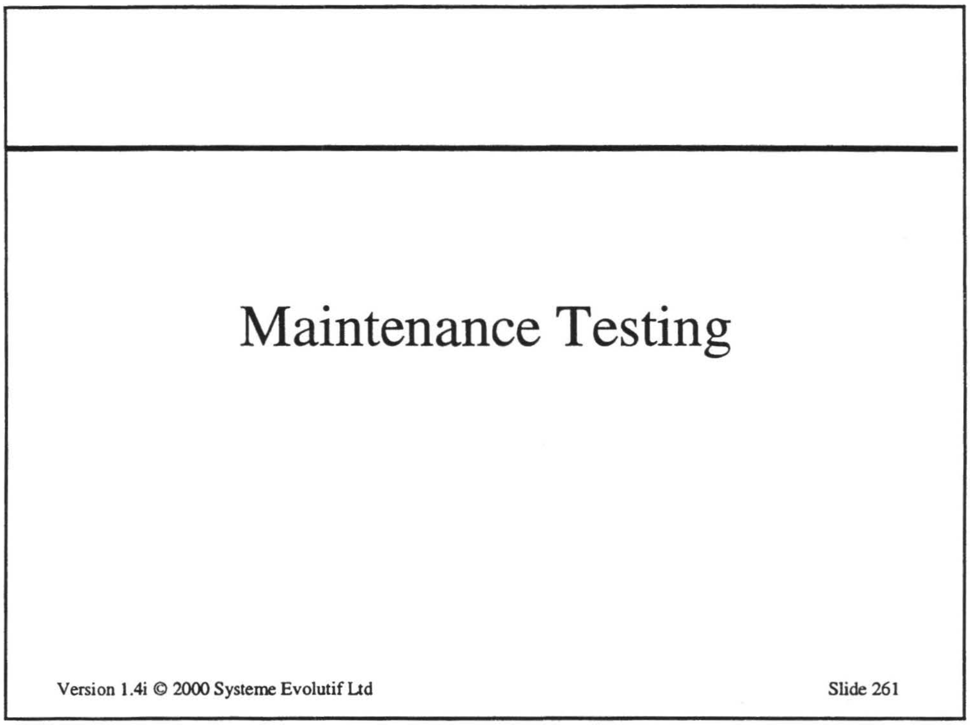
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 259



Typical test scenarios (recovery from failure)

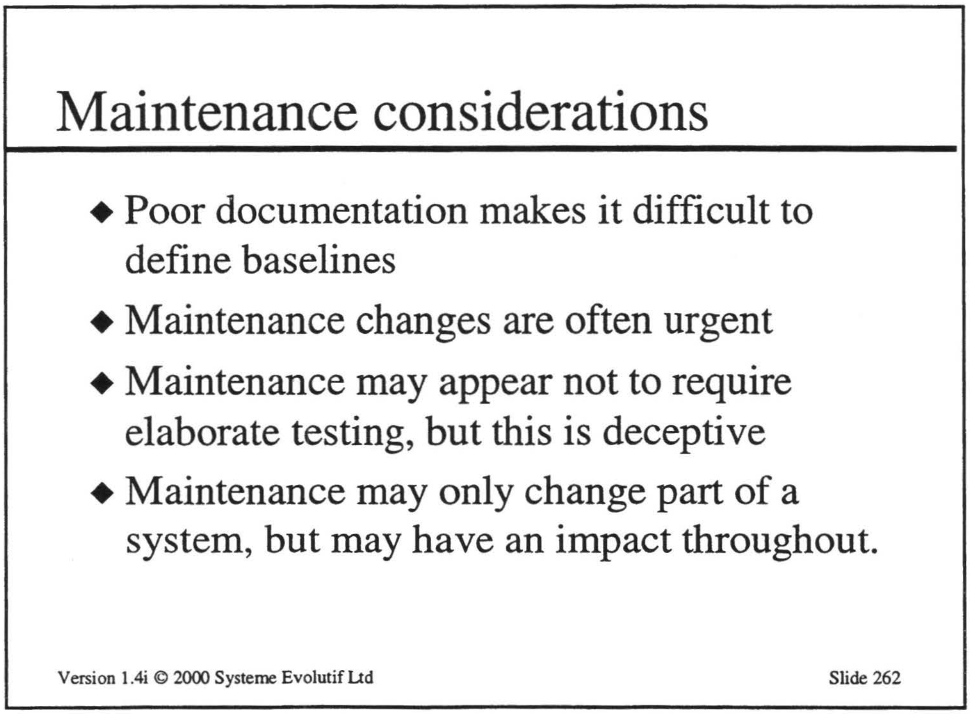
- ◆ While entering transactions into the database, bring the machine down by causing (or simulating) a machine crash
- ◆ Reboot the machine and demonstrate by means of query or reporting, that the database has recovered the transactions committed up to the point of failure.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 260



Maintenance Testing

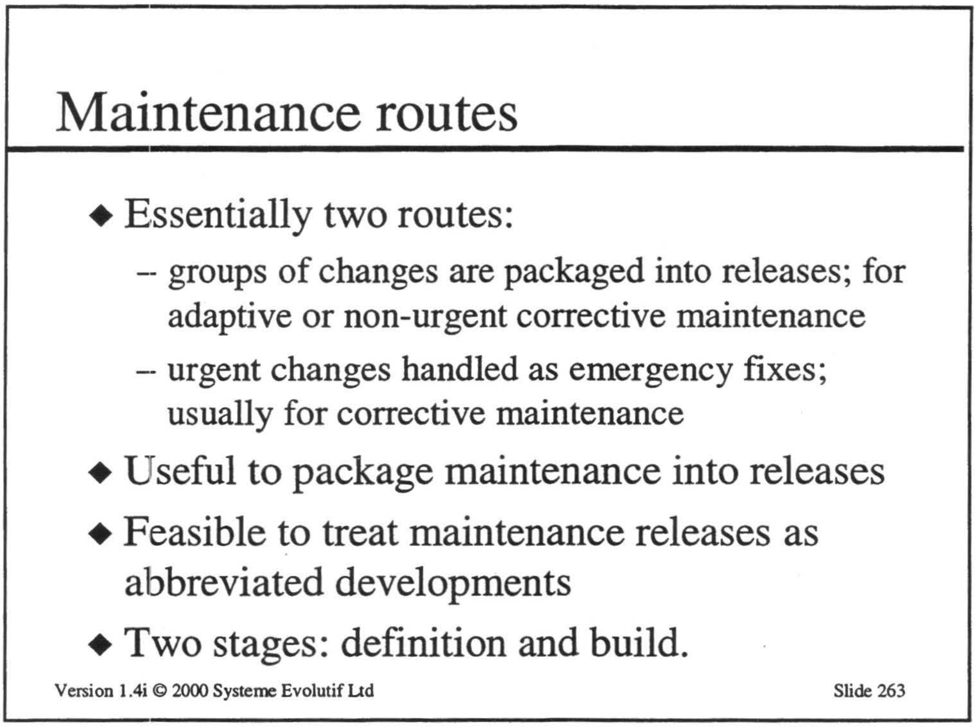
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 261



Maintenance considerations

- ◆ Poor documentation makes it difficult to define baselines
- ◆ Maintenance changes are often urgent
- ◆ Maintenance may appear not to require elaborate testing, but this is deceptive
- ◆ Maintenance may only change part of a system, but may have an impact throughout.

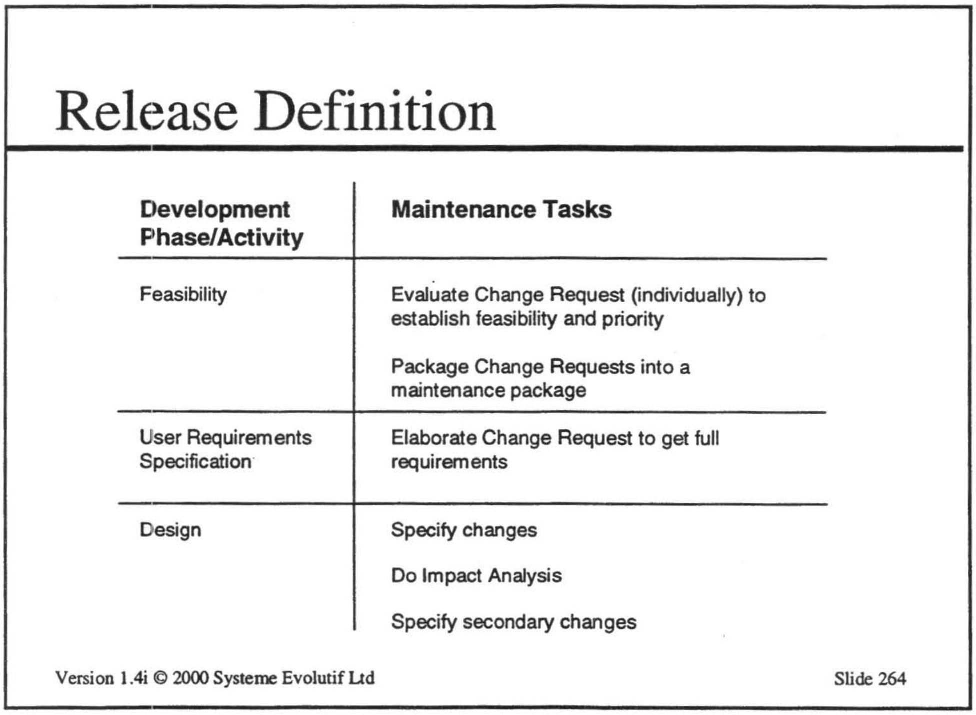
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 262



Maintenance routes

- Essentially two routes:
  - groups of changes are packaged into releases; for adaptive or non-urgent corrective maintenance
  - urgent changes handled as emergency fixes; usually for corrective maintenance
- Useful to package maintenance into releases
- Feasible to treat maintenance releases as abbreviated developments
- Two stages: definition and build.

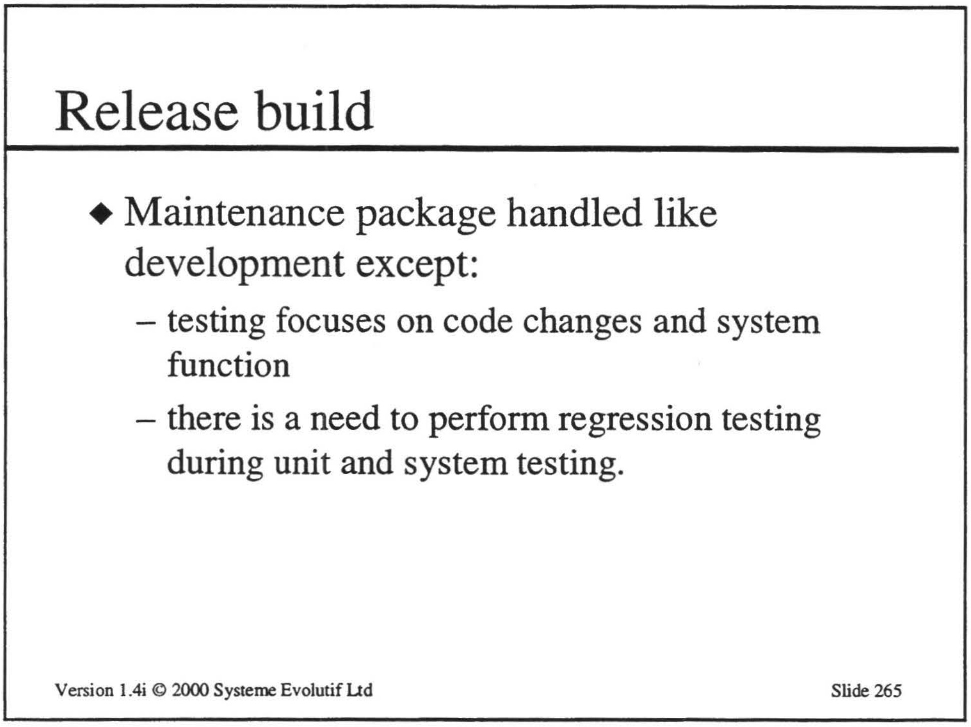
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 263



Release Definition

Development Phase/Activity	**Maintenance Tasks**
Feasibility	Evaluate Change Request (individually) to establish feasibility and priority Package Change Requests into a maintenance package
User Requirements Specification	Elaborate Change Request to get full requirements
Design	Specify changes Do Impact Analysis Specify secondary changes

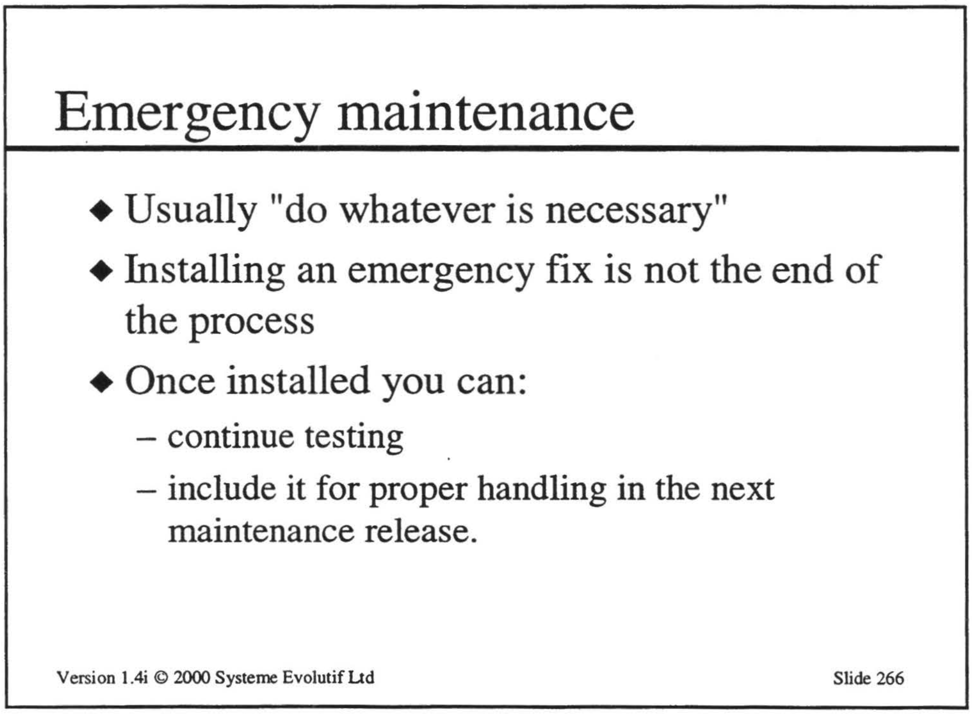
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 264



Release build

- ◆ Maintenance package handled like development except:
  - – testing focuses on code changes and system function
  - – there is a need to perform regression testing during unit and system testing.

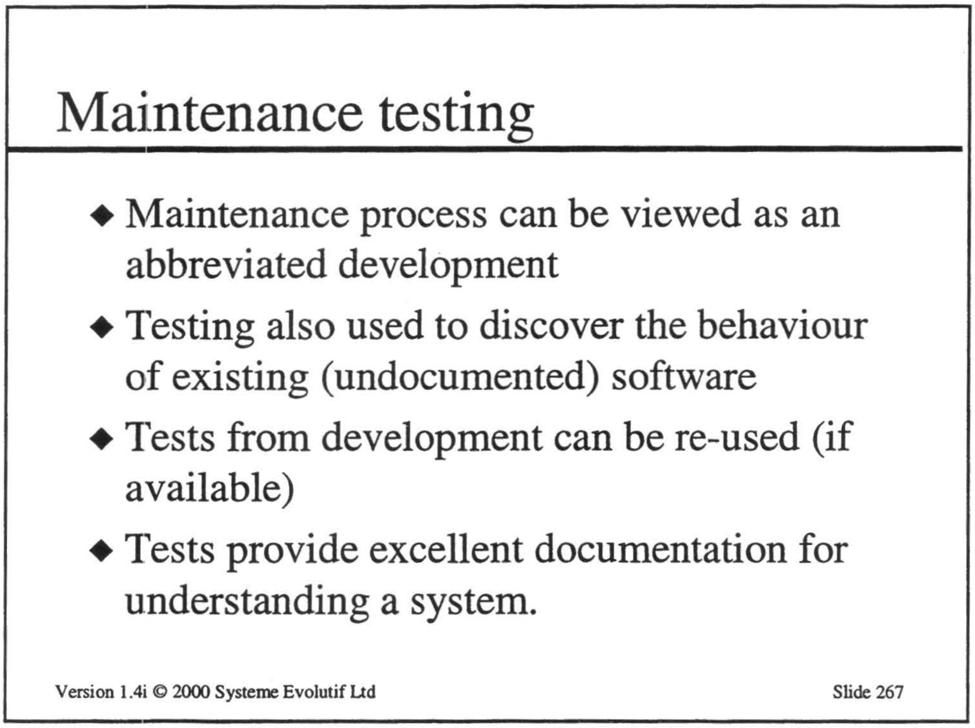
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 265



Emergency maintenance

- ◆ Usually "do whatever is necessary"
- ◆ Installing an emergency fix is not the end of the process
- ◆ Once installed you can:
  - – continue testing
  - – include it for proper handling in the next maintenance release.

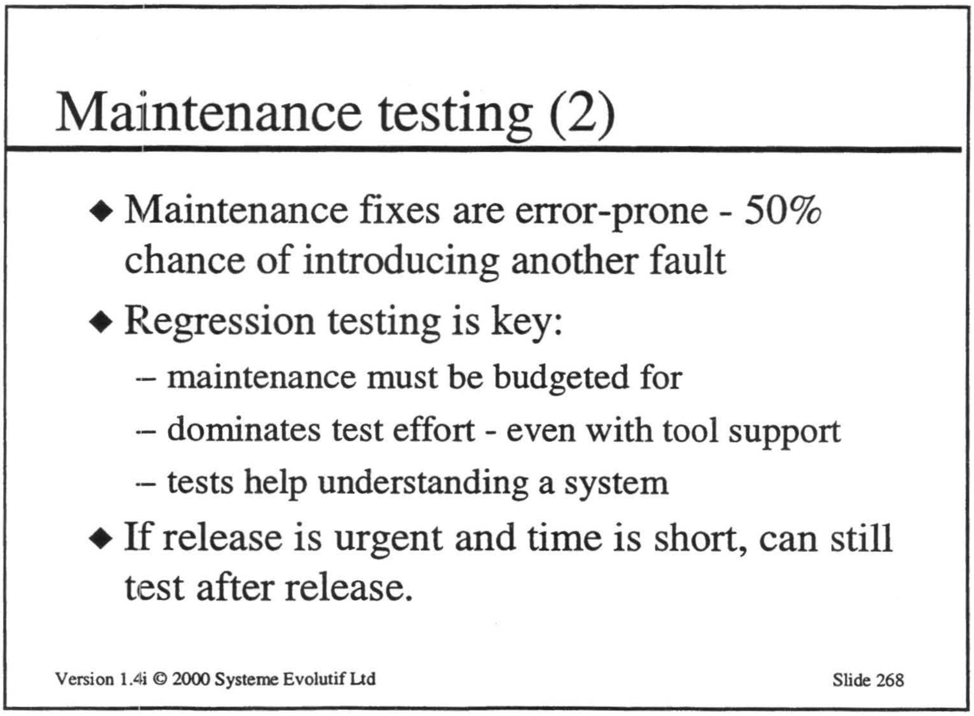
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 266



Maintenance testing

- ◆ Maintenance process can be viewed as an abbreviated development
- ◆ Testing also used to discover the behaviour of existing (undocumented) software
- ◆ Tests from development can be re-used (if available)
- ◆ Tests provide excellent documentation for understanding a system.

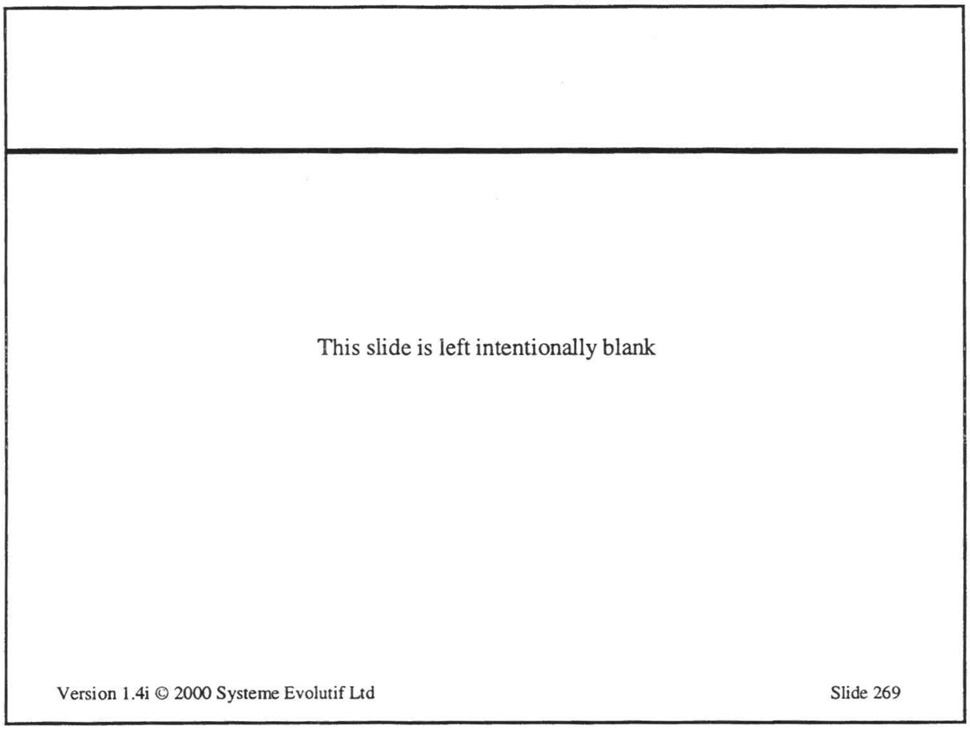
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 267



Maintenance testing (2)

- ◆ Maintenance fixes are error-prone - 50% chance of introducing another fault
- ◆ Regression testing is key:
  - – maintenance must be budgeted for
  - – dominates test effort - even with tool support
  - – tests help understanding a system
- ◆ If release is urgent and time is short, can still test after release.

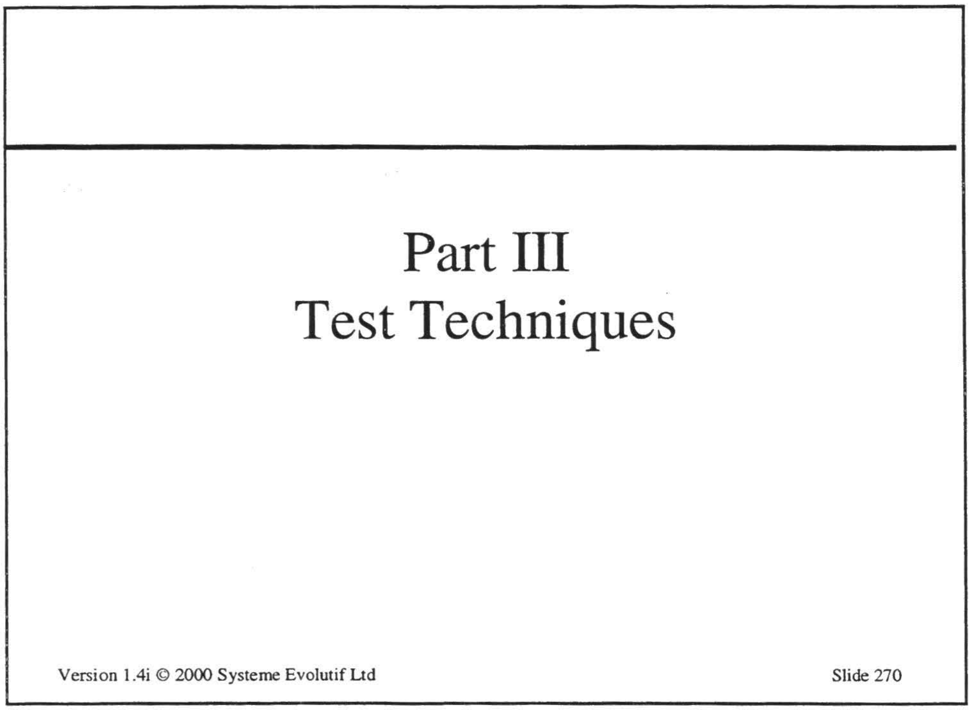
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 268



This slide is left intentionally blank

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 269



Part III
Test Techniques

Version 1.4i © 2000 Systeme Evolutif Ltd

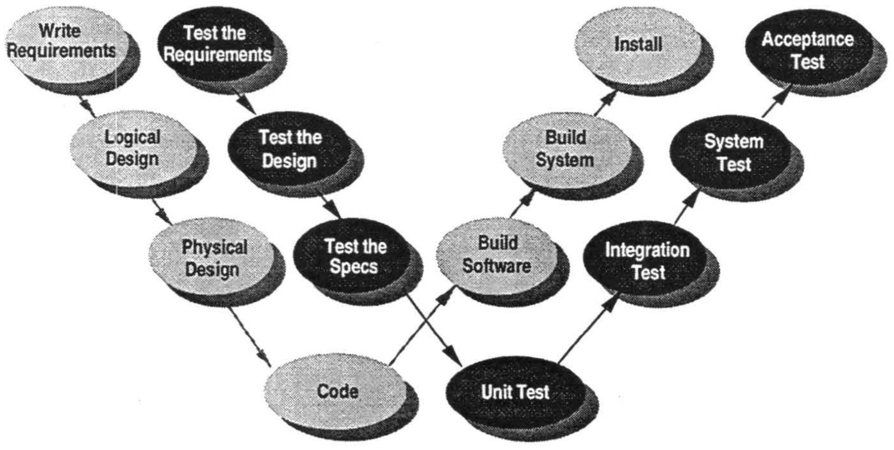
Slide 270

Test Techniques and the Lifecycle

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 271

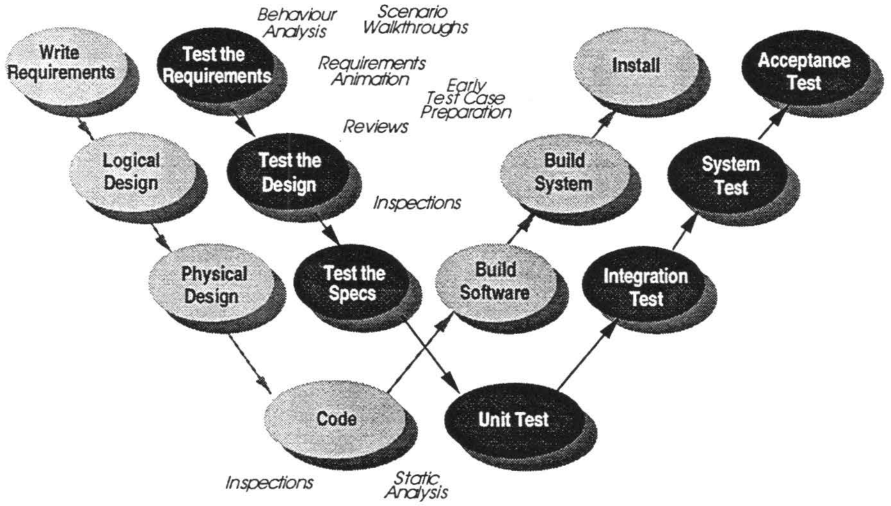
Testing throughout the life cycle: the W model



Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 272

Static test techniques



Version 1.4i © 2000 Systeme Evolutif Ltd Slide 273

Comparative testing efficiencies

Testing Type	***Efficiency (faults found / hour)***
Operational Use	.21
Black Box	.282
White Box	.322
Reading / Inspections	1.057

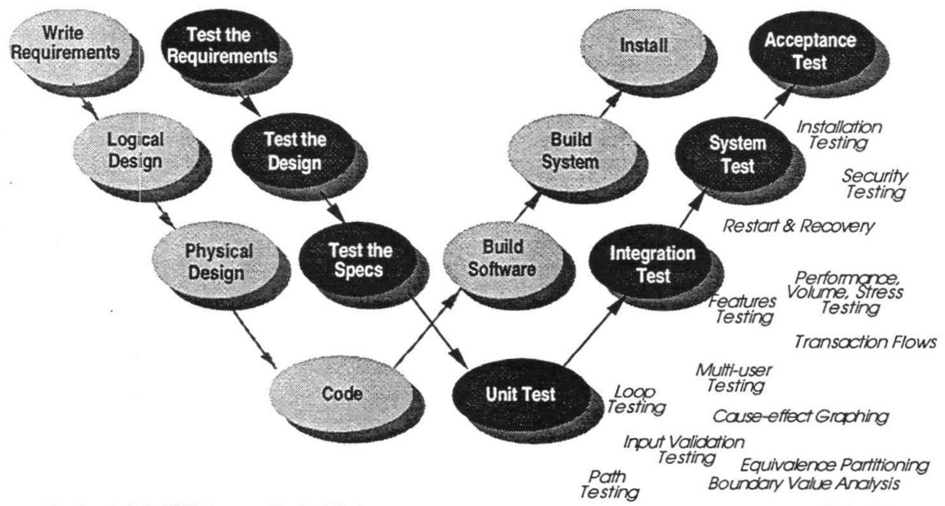
"Practical Software Metrics for Project Management and Process Improvement", R. B. Grady, Prentice-Hall, 1992

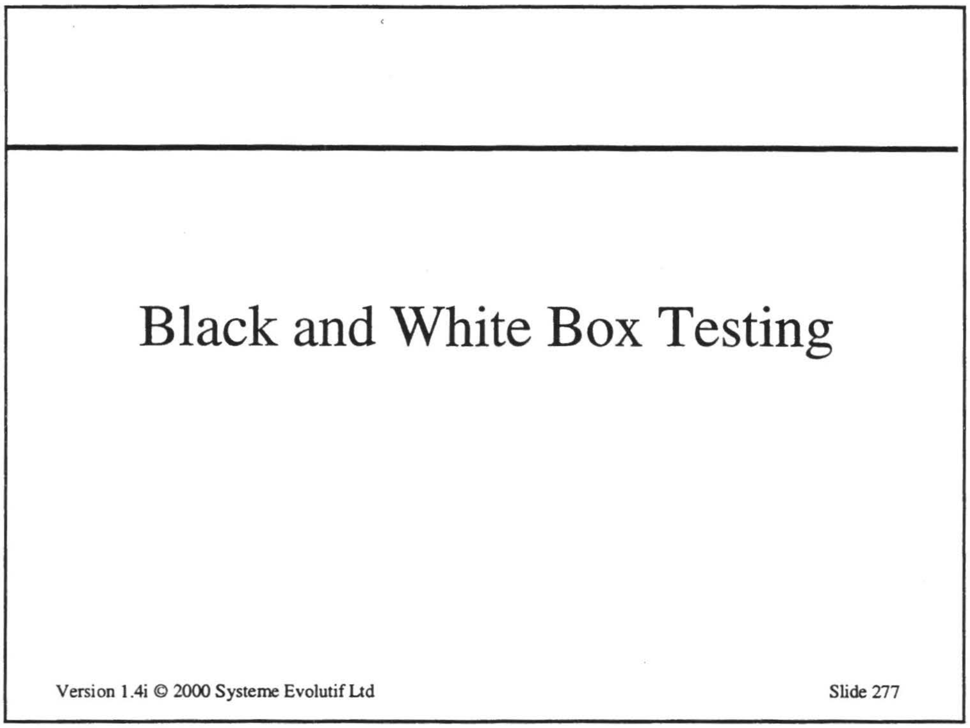
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 274

Static analysis

- Automated code inspection e.g.
  - Compiler
  - Lint
  - "Deep-flow" static analysis

Dynamic test techniques

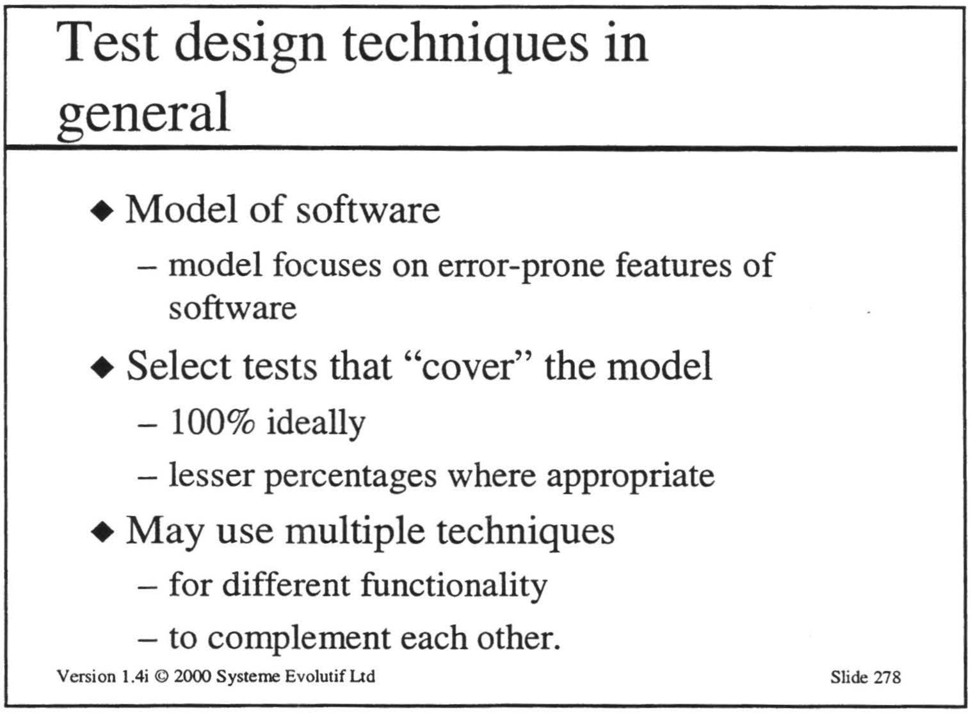




Black and White Box Testing

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 277

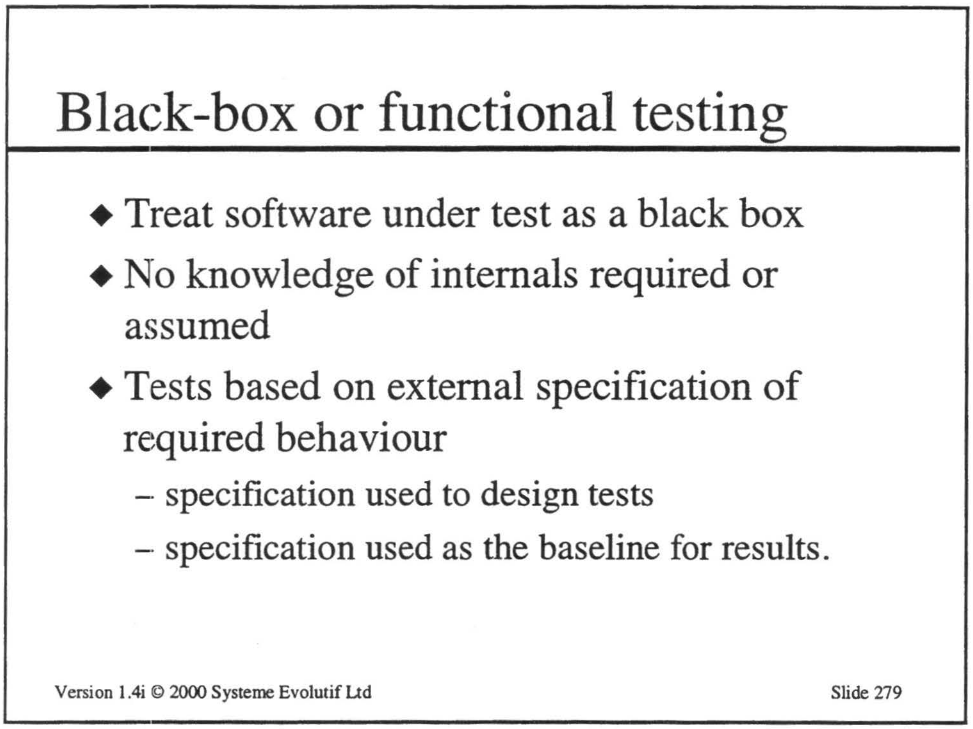


Test design techniques in general

- Model of software
  - model focuses on error-prone features of software
- Select tests that “cover” the model
  - 100% ideally
  - lesser percentages where appropriate
- May use multiple techniques
  - for different functionality
  - to complement each other.

Version 1.4i © 2000 Systeme Evolutif Ltd

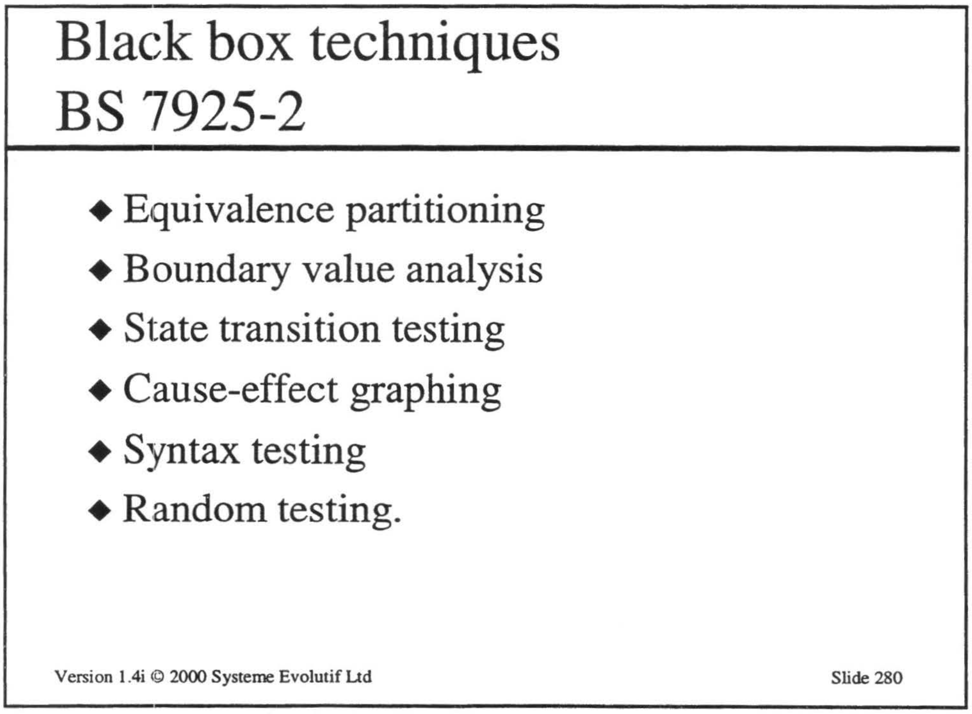
Slide 278



Black-box or functional testing

- Treat software under test as a black box
- No knowledge of internals required or assumed
- Tests based on external specification of required behaviour
  - specification used to design tests
  - specification used as the baseline for results.

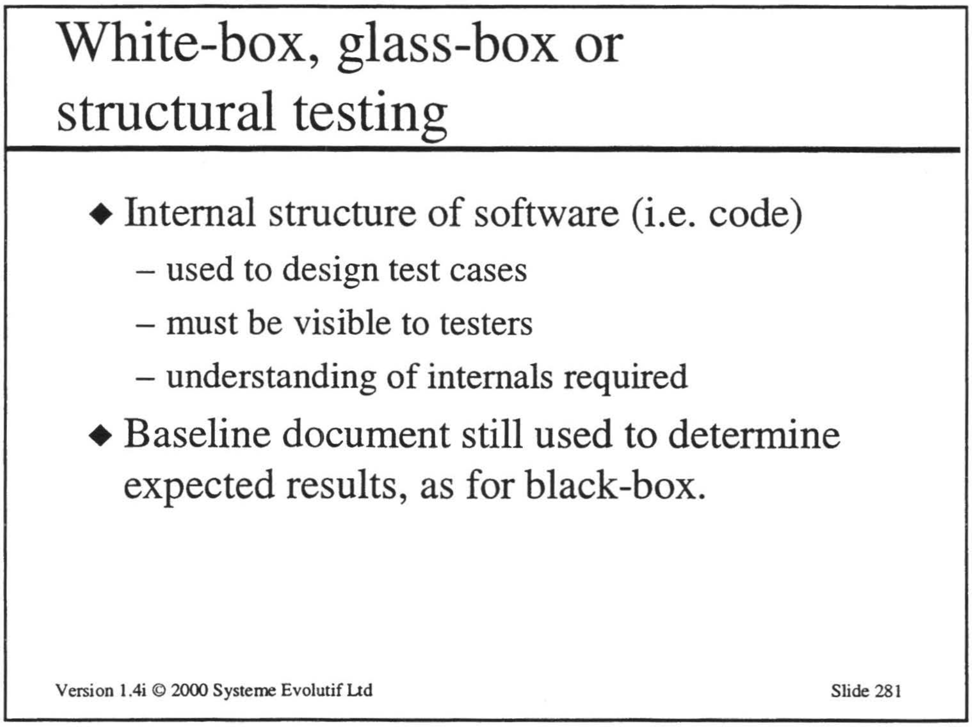
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 279



Black box techniques BS 7925-2

- Equivalence partitioning
- Boundary value analysis
- State transition testing
- Cause-effect graphing
- Syntax testing
- Random testing.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 280

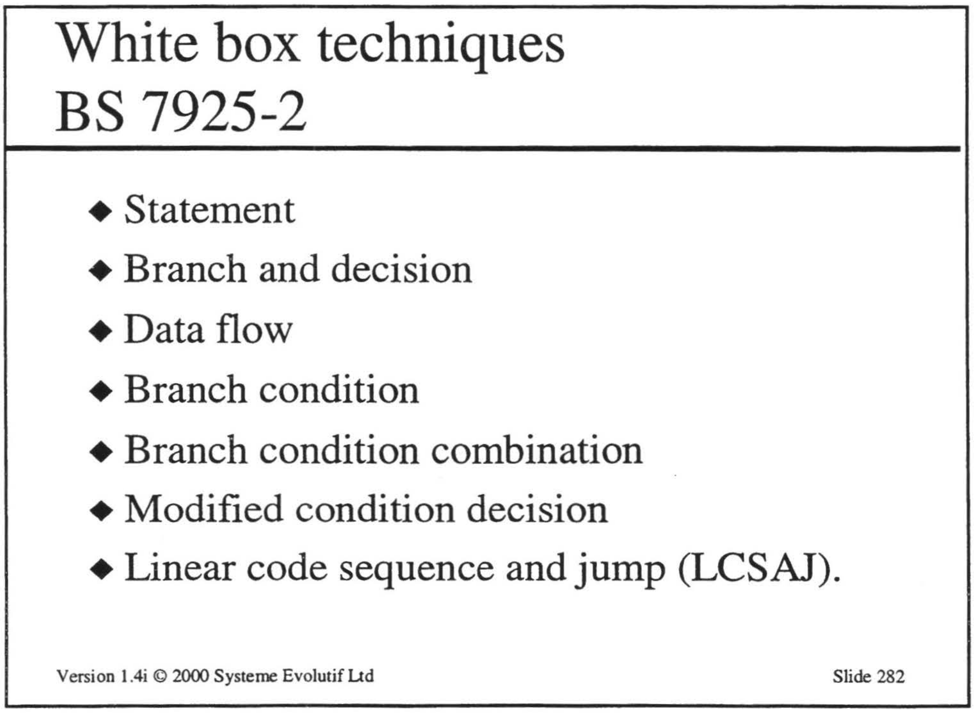


White-box, glass-box or structural testing

- Internal structure of software (i.e. code)
  - used to design test cases
  - must be visible to testers
  - understanding of internals required
- Baseline document still used to determine expected results, as for black-box.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 281

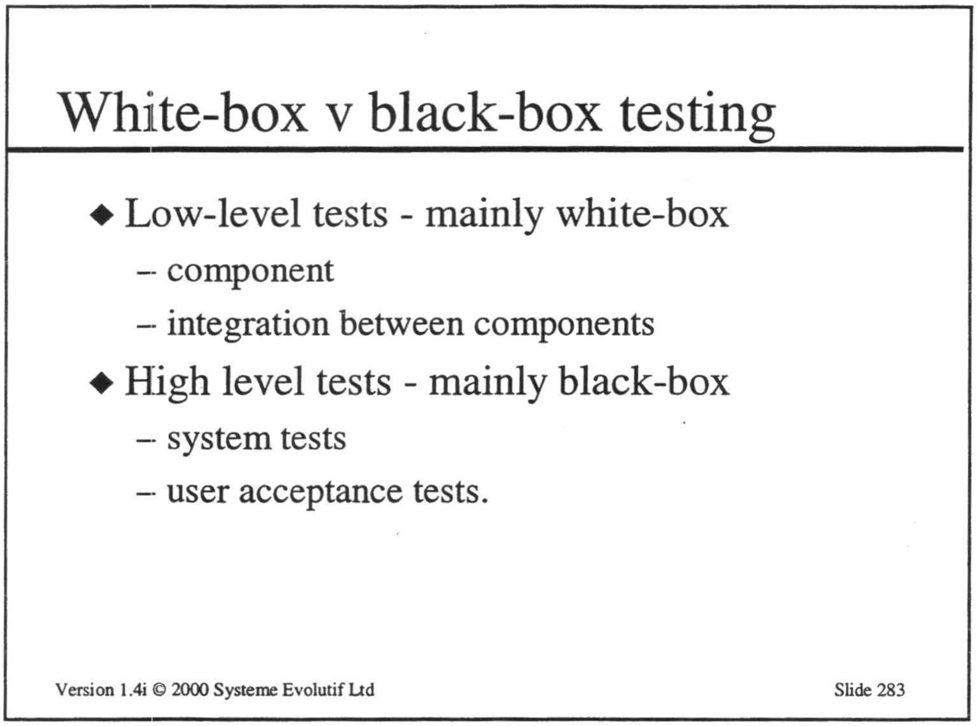


White box techniques BS 7925-2

- Statement
- Branch and decision
- Data flow
- Branch condition
- Branch condition combination
- Modified condition decision
- Linear code sequence and jump (LCSAJ).

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 282

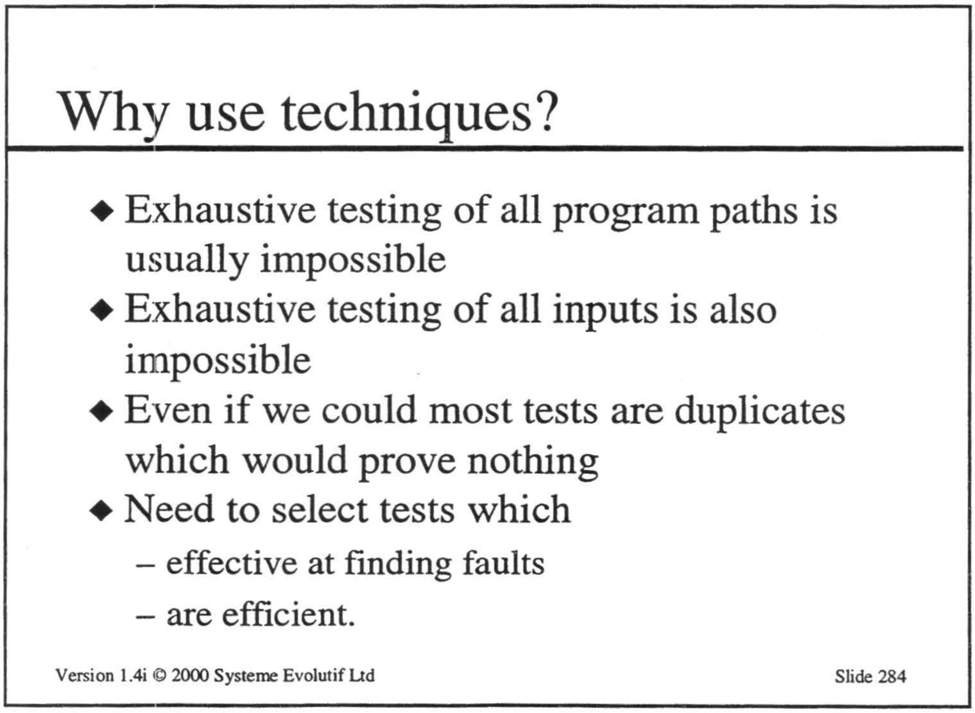


White-box v black-box testing

- Low-level tests - mainly white-box
  - component
  - integration between components
- High level tests - mainly black-box
  - system tests
  - user acceptance tests.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 283

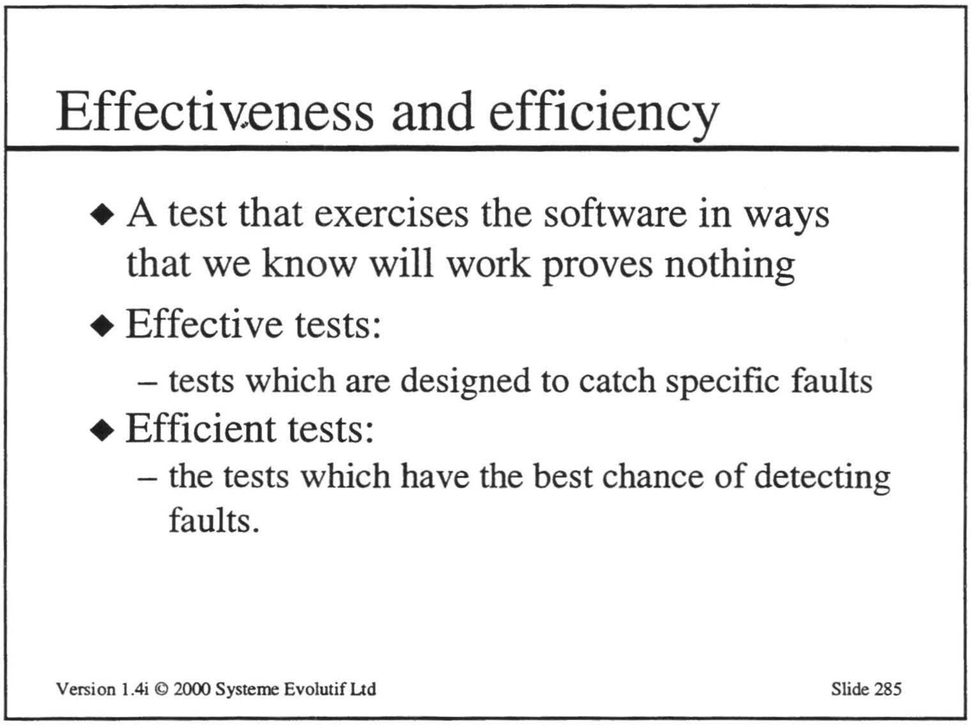


Why use techniques?

- Exhaustive testing of all program paths is usually impossible
- Exhaustive testing of all inputs is also impossible
- Even if we could most tests are duplicates which would prove nothing
- Need to select tests which
  - effective at finding faults
  - are efficient.

Version 1.4i © 2000 Systeme Evolutif Ltd

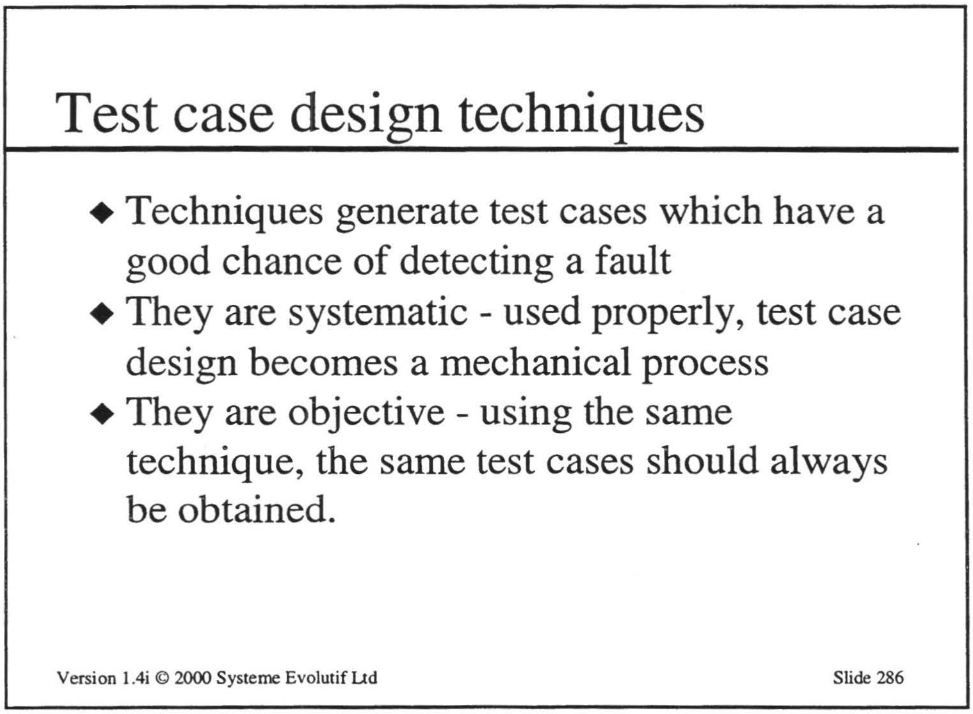
Slide 284



Effectiveness and efficiency

- ◆ A test that exercises the software in ways that we know will work proves nothing
- ◆ Effective tests:
  - – tests which are designed to catch specific faults
- ◆ Efficient tests:
  - – the tests which have the best chance of detecting faults.

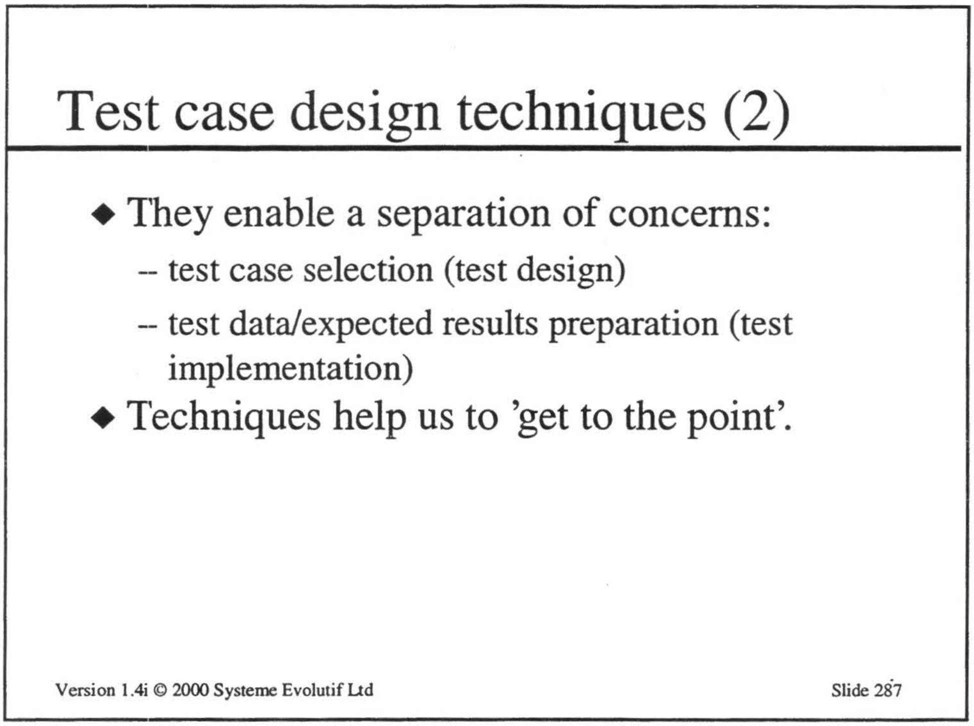
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 285



Test case design techniques

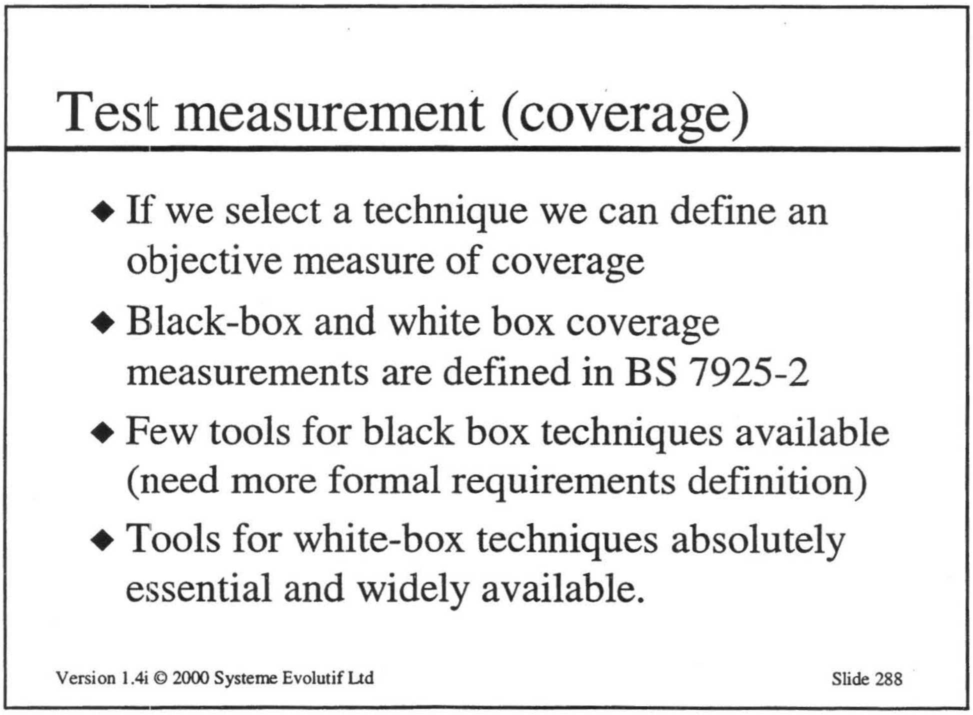
- ◆ Techniques generate test cases which have a good chance of detecting a fault
- ◆ They are systematic - used properly, test case design becomes a mechanical process
- ◆ They are objective - using the same technique, the same test cases should always be obtained.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 286



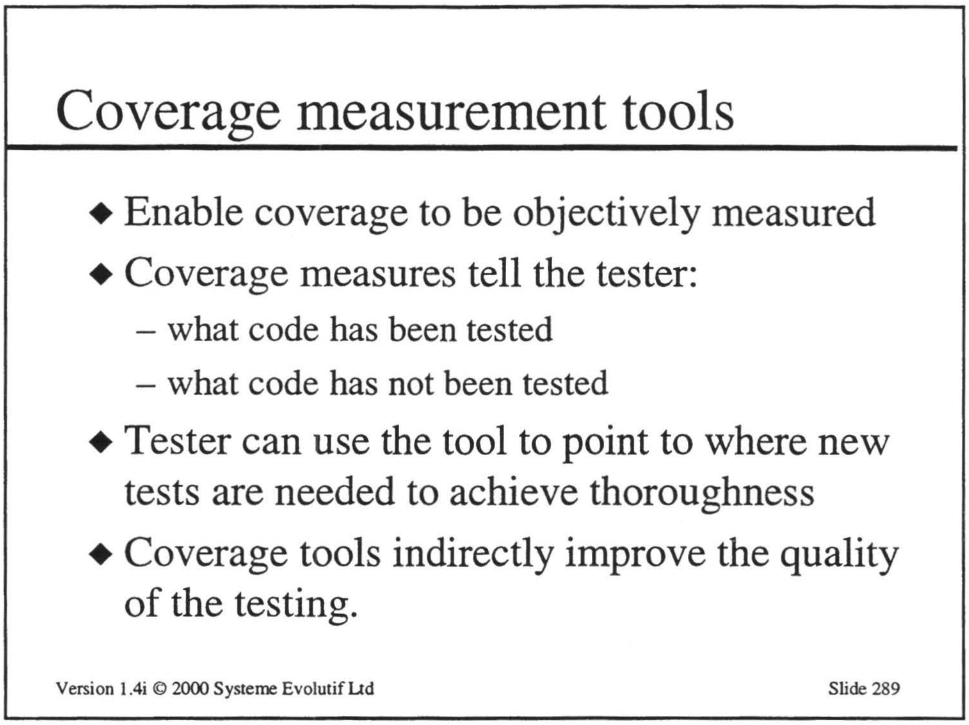
Test case design techniques (2)

- They enable a separation of concerns:
  - test case selection (test design)
  - test data/expected results preparation (test implementation)
- Techniques help us to 'get to the point'.



Test measurement (coverage)

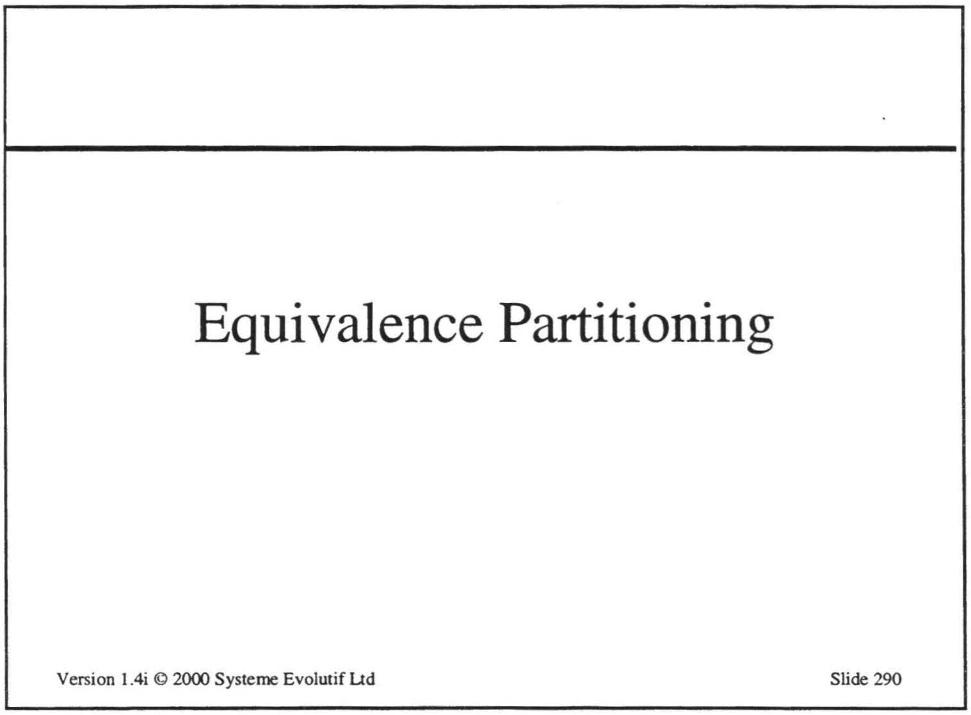
- If we select a technique we can define an objective measure of coverage
- Black-box and white box coverage measurements are defined in BS 7925-2
- Few tools for black box techniques available (need more formal requirements definition)
- Tools for white-box techniques absolutely essential and widely available.



Coverage measurement tools

- Enable coverage to be objectively measured
- Coverage measures tell the tester:
  - what code has been tested
  - what code has not been tested
- Tester can use the tool to point to where new tests are needed to achieve thoroughness
- Coverage tools indirectly improve the quality of the testing.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 289



Equivalence Partitioning

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 290

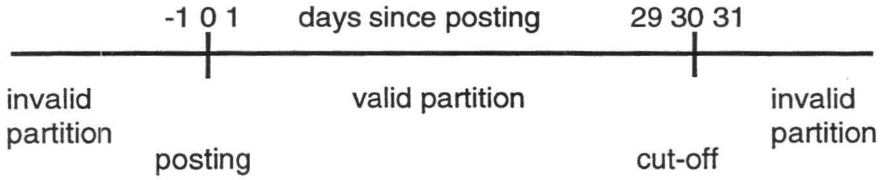
Equivalence partitioning

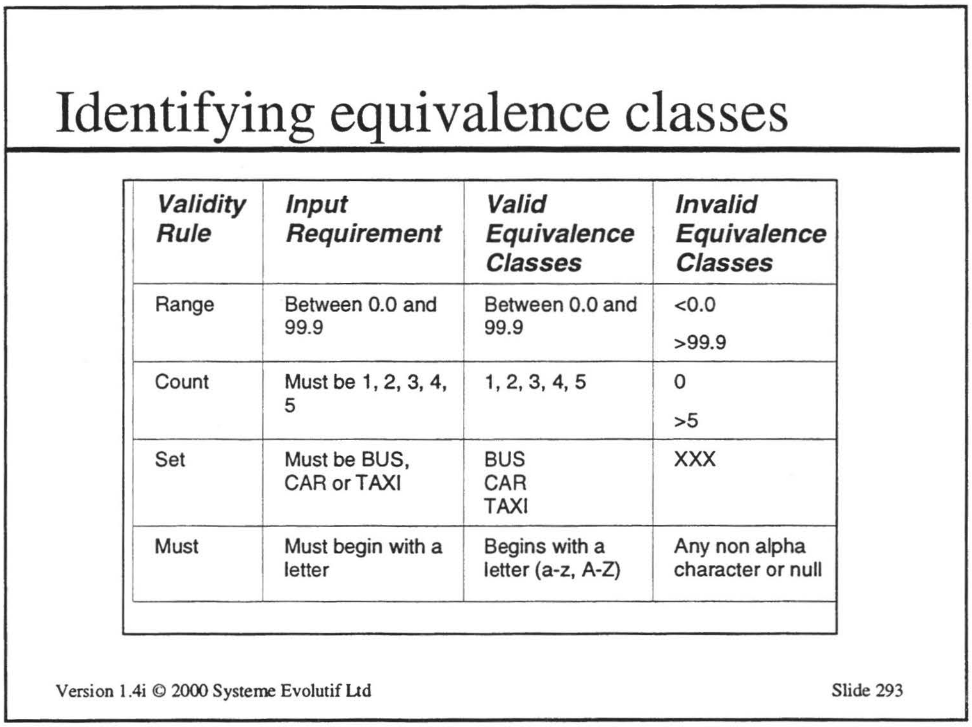
- Software generalises
- Groups of inputs will be processed in the same way
- Can choose a representative input from the group
- We do this intuitively already to select test cases
- Equivalence partitioning formalises this common technique

Equivalence partitioning example

- Requirement

Contestants have 30 days from date of posting form to return entry form

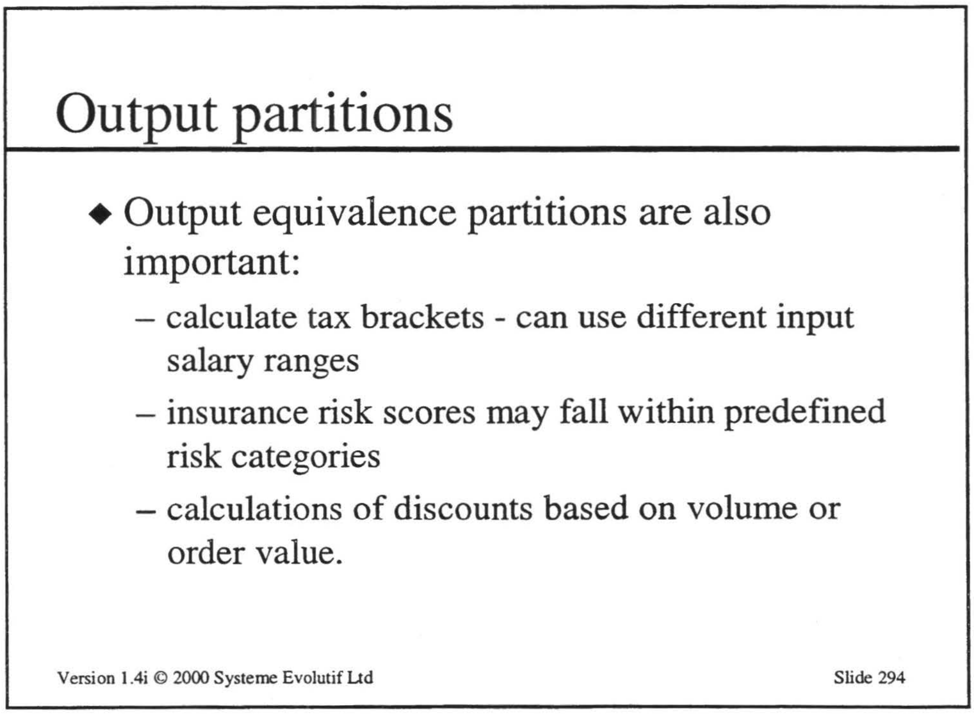




Identifying equivalence classes

Validity Rule	***Input Requirement***	***Valid Equivalence Classes***	***Invalid Equivalence Classes***
Range	Between 0.0 and 99.9	Between 0.0 and 99.9	<0.0 >99.9
Count	Must be 1, 2, 3, 4, 5	1, 2, 3, 4, 5	0 >5
Set	Must be BUS, CAR or TAXI	BUS CAR TAXI	XXX
Must	Must begin with a letter	Begins with a letter (a-z, A-Z)	Any non alpha character or null

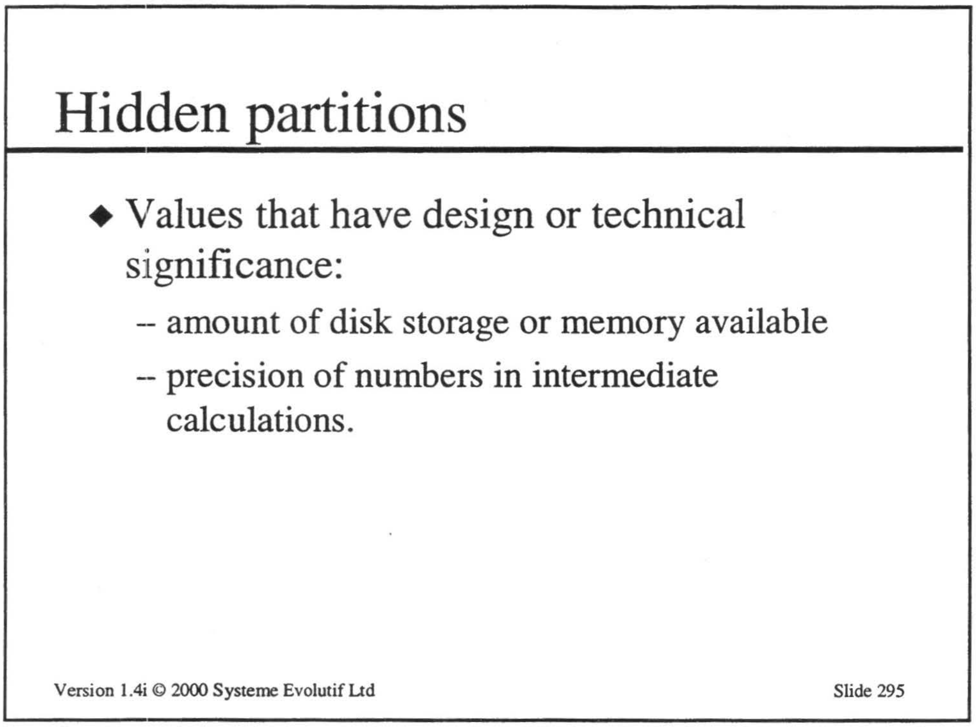
Version 1.4i © 2000 Systeme Evolutif Ltd   Slide 293



Output partitions

- Output equivalence partitions are also important:
  - calculate tax brackets - can use different input salary ranges
  - insurance risk scores may fall within predefined risk categories
  - calculations of discounts based on volume or order value.

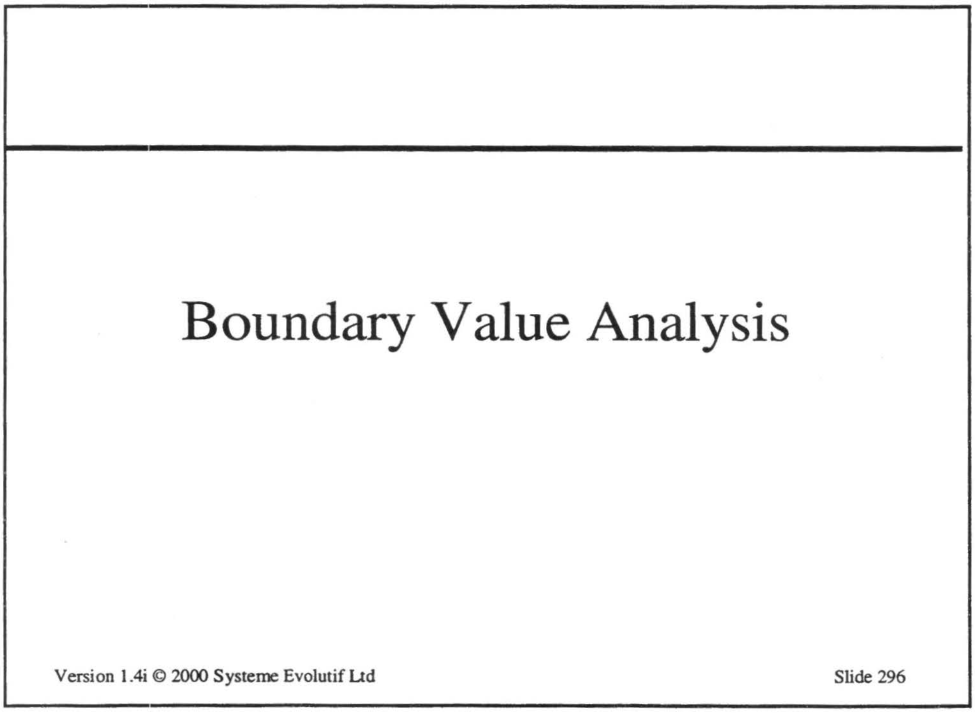
Version 1.4i © 2000 Systeme Evolutif Ltd   Slide 294



Hidden partitions

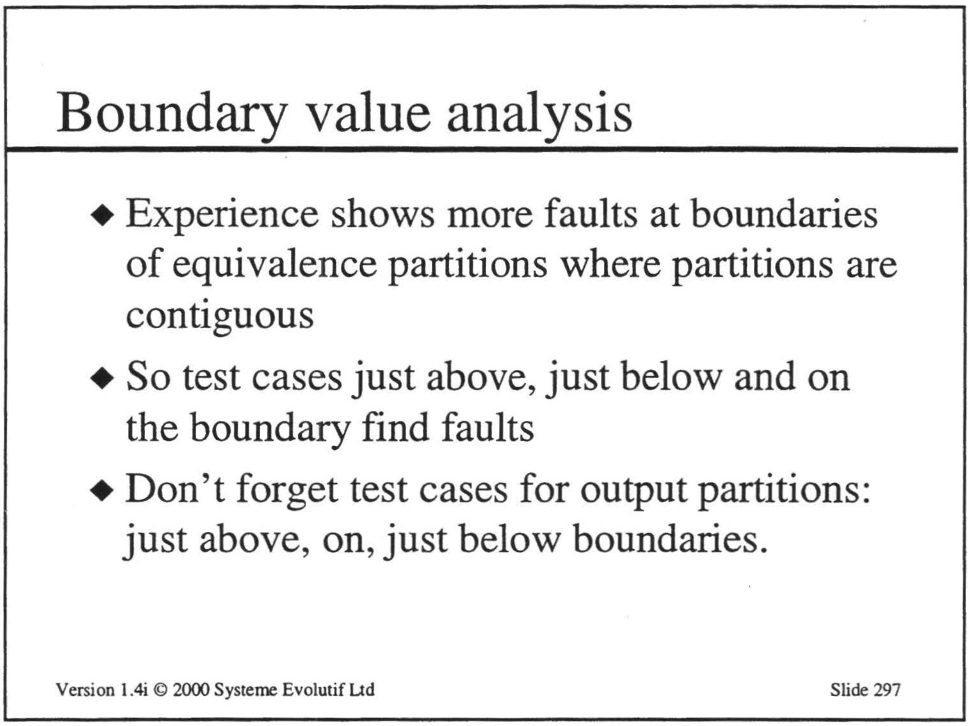
- Values that have design or technical significance:
  - amount of disk storage or memory available
  - precision of numbers in intermediate calculations.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 295



Boundary Value Analysis

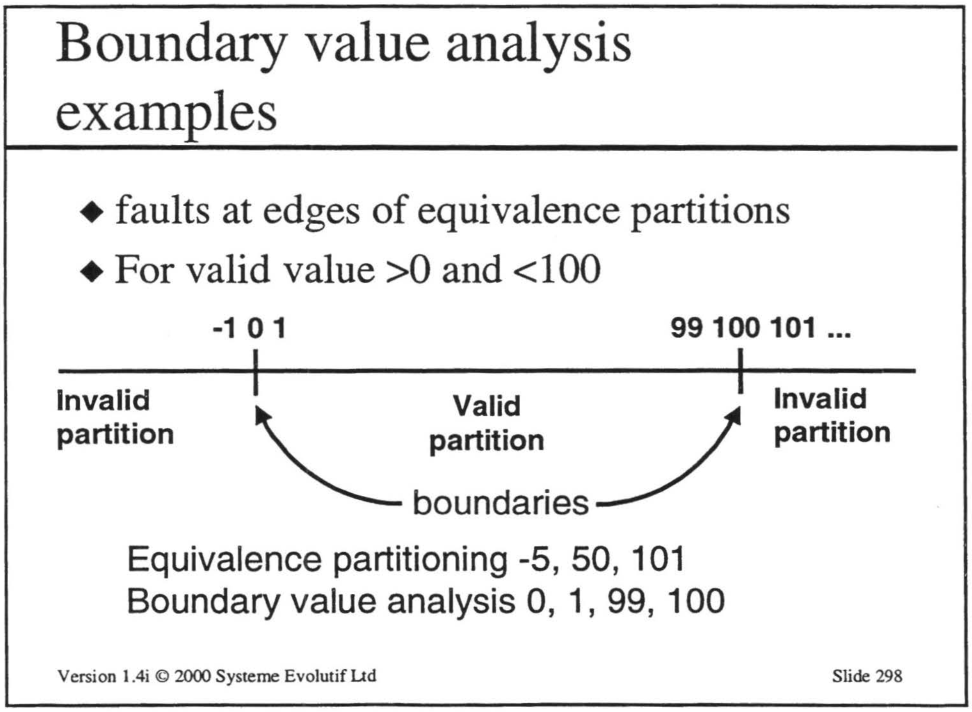
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 296



Boundary value analysis

- Experience shows more faults at boundaries of equivalence partitions where partitions are contiguous
- So test cases just above, just below and on the boundary find faults
- Don’t forget test cases for output partitions: just above, on, just below boundaries.

Version 1.4i © 2000 Systeme Evolutif Ltd — Slide 297



Boundary value analysis examples

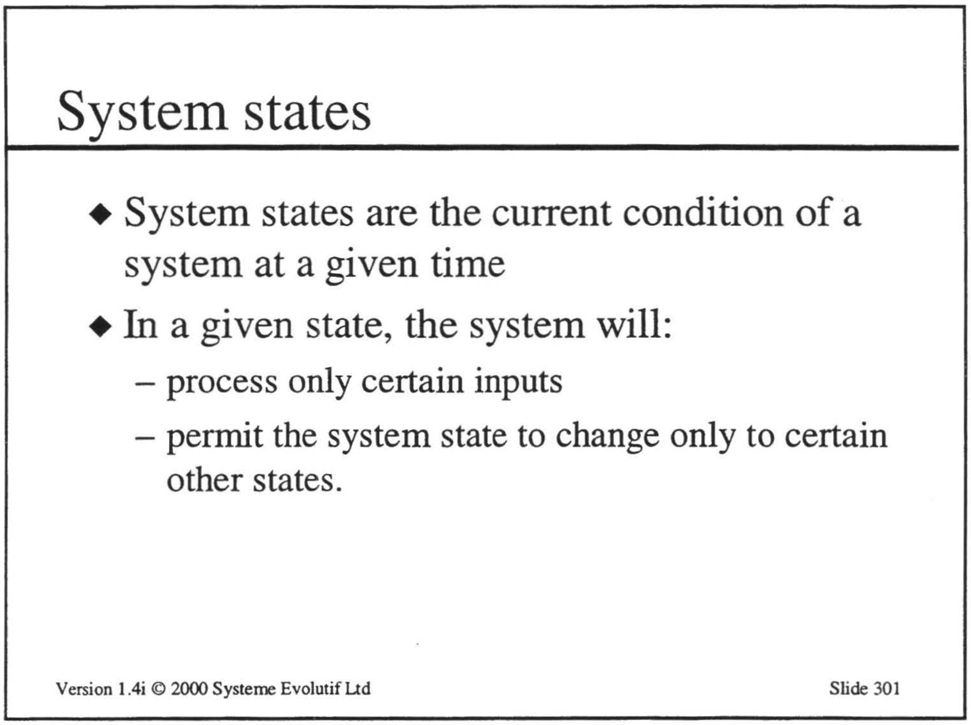
- faults at edges of equivalence partitions
- For valid value >0 and <100

Equivalence partitioning -5, 50, 101
Boundary value analysis 0, 1, 99, 100

Version 1.4i © 2000 Systeme Evolutif Ltd — Slide 298

Exercise 1

State Transition Testing



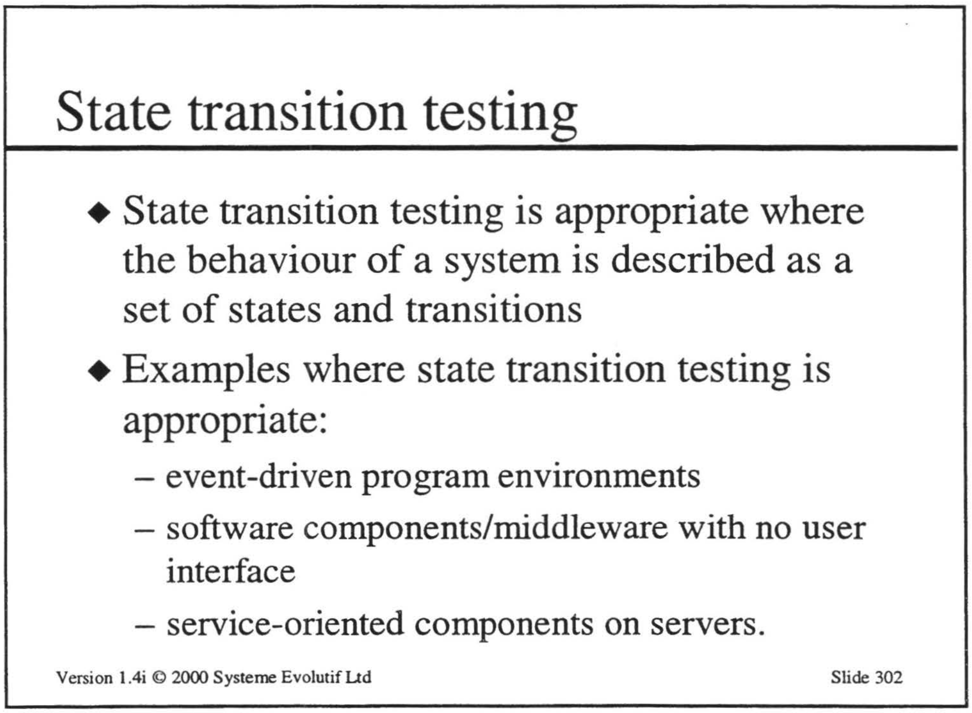
System states

- System states are the current condition of a system at a given time
- In a given state, the system will:
  - process only certain inputs
  - permit the system state to change only to certain other states.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 301

3

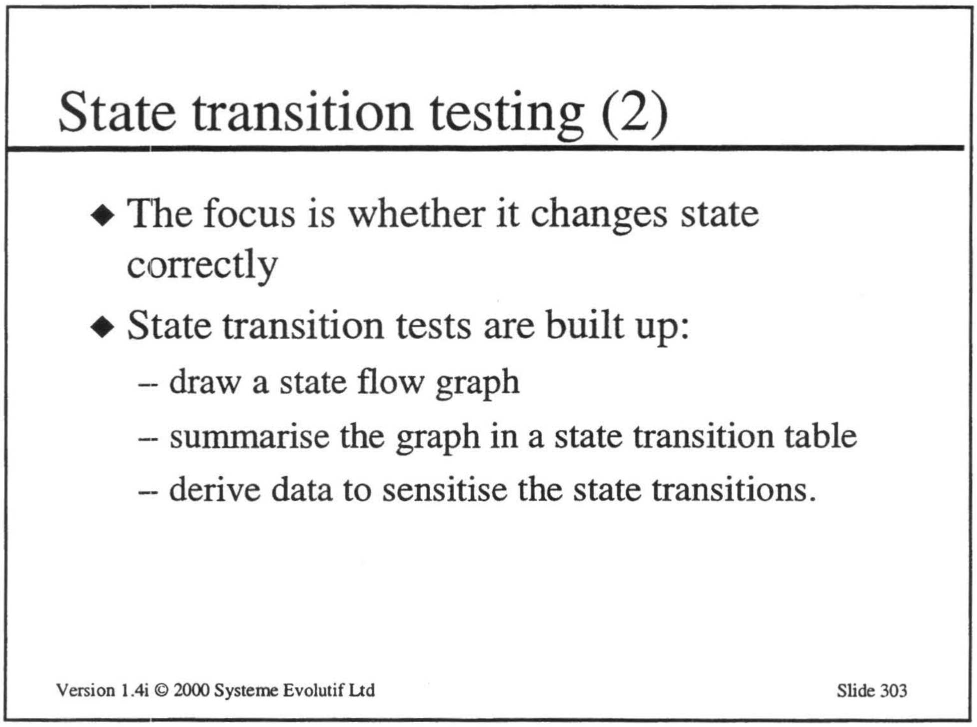


State transition testing

- State transition testing is appropriate where the behaviour of a system is described as a set of states and transitions
- Examples where state transition testing is appropriate:
  - event-driven program environments
  - software components/middleware with no user interface
  - service-oriented components on servers.

Version 1.4i © 2000 Systeme Evolutif Ltd

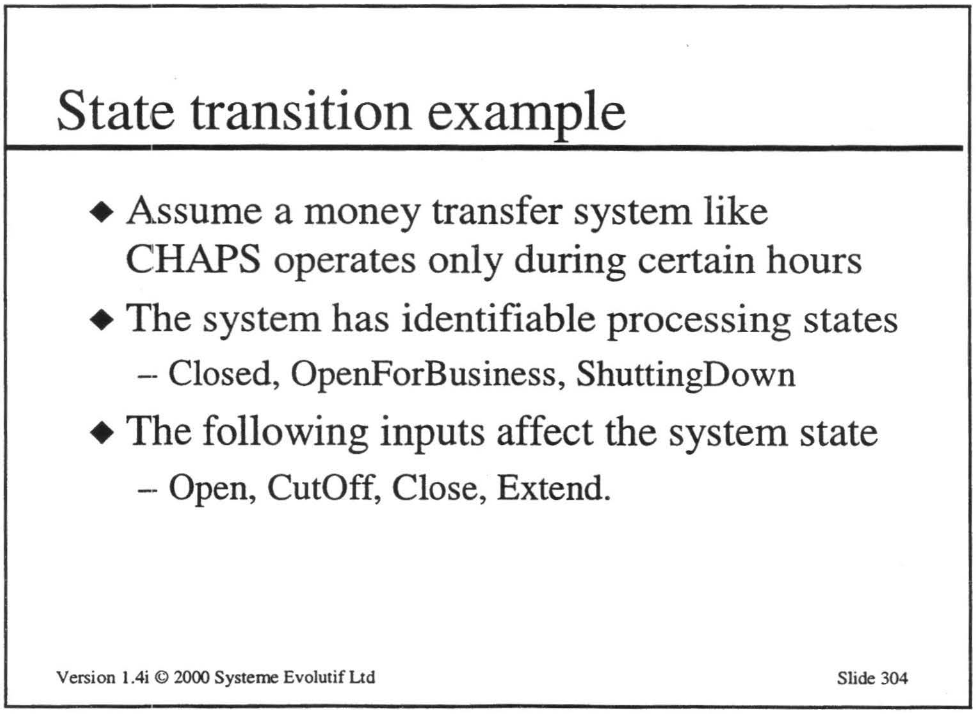
Slide 302



State transition testing (2)

- The focus is whether it changes state correctly
- State transition tests are built up:
  - draw a state flow graph
  - summarise the graph in a state transition table
  - derive data to sensitise the state transitions.

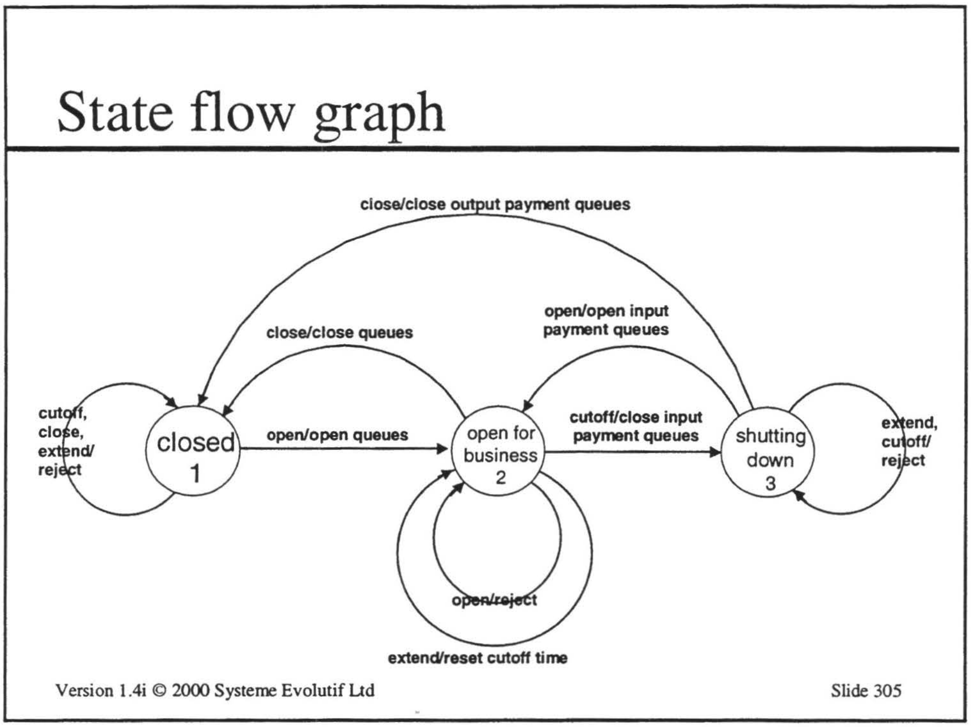
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 303



State transition example

- Assume a money transfer system like CHAPS operates only during certain hours
- The system has identifiable processing states
  - Closed, OpenForBusiness, ShuttingDown
- The following inputs affect the system state
  - Open, CutOff, Close, Extend.

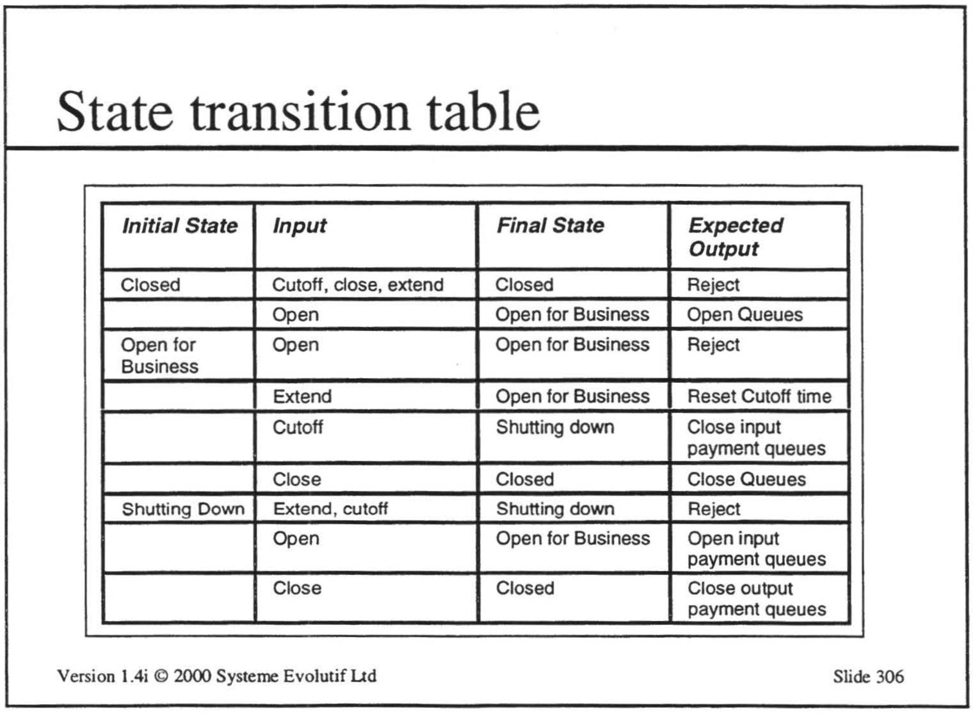
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 304



State flow graph

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 305

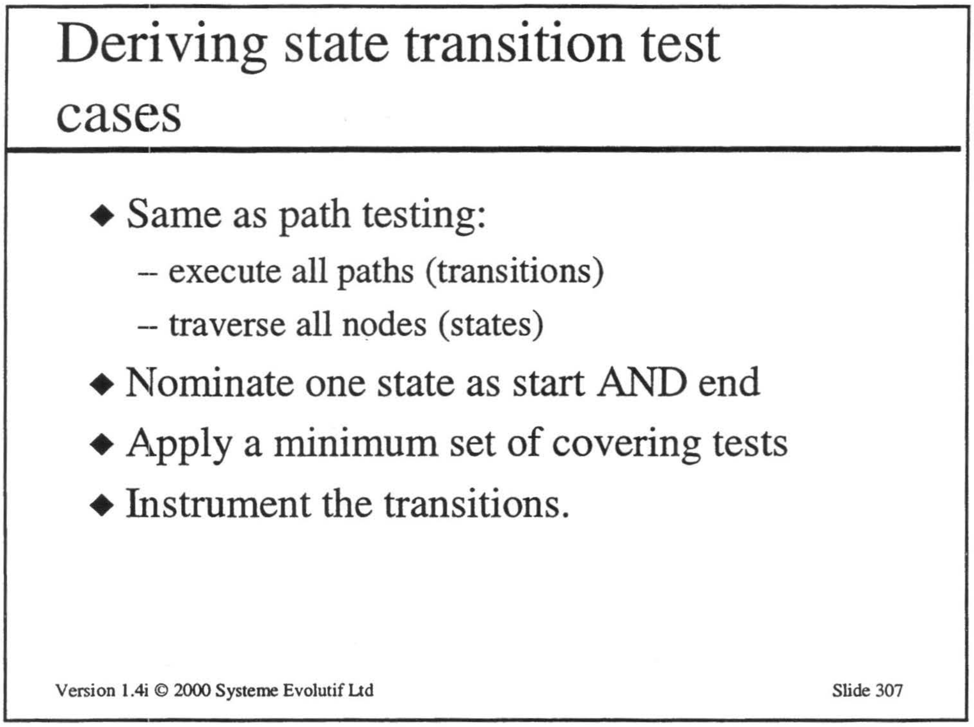


State transition table

Initial State	***Input***	***Final State***	***Expected Output***
Closed	Cutoff, close, extend	Closed	Reject
	Open	Open for Business	Open Queues
Open for Business	Open	Open for Business	Reject
	Extend	Open for Business	Reset Cutoff time
	Cutoff	Shutting down	Close input payment queues
	Close	Closed	Close Queues
Shutting Down	Extend, cutoff	Shutting down	Reject
	Open	Open for Business	Open input payment queues
	Close	Closed	Close output payment queues

Version 1.4i © 2000 Systeme Evolutif Ltd

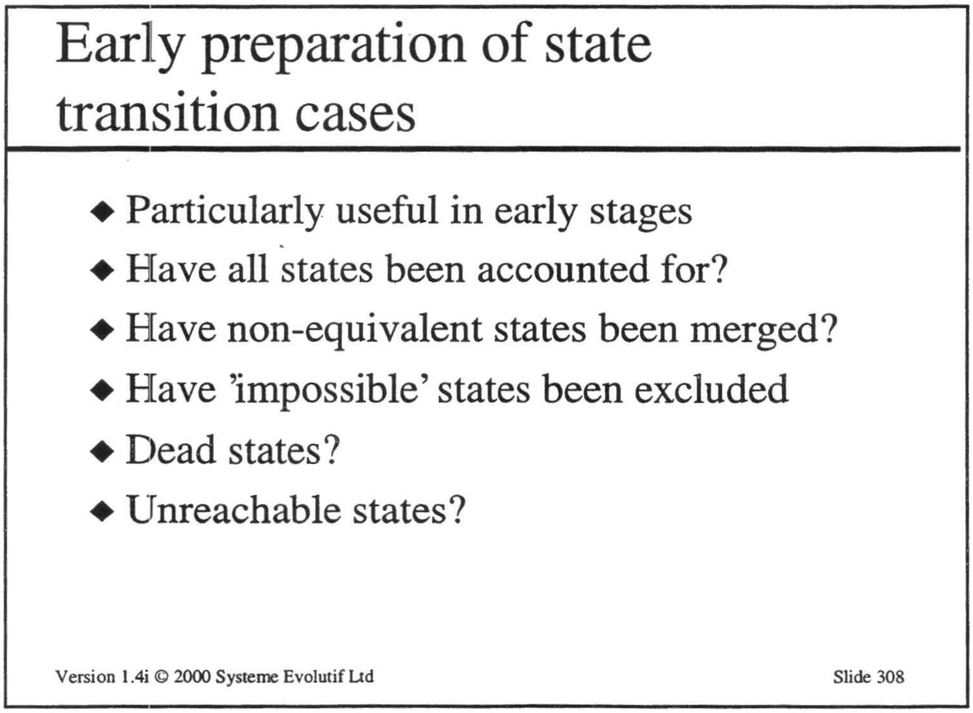
Slide 306



Deriving state transition test cases

- Same as path testing:
  - -- execute all paths (transitions)
  - -- traverse all nodes (states)
- Nominate one state as start AND end
- Apply a minimum set of covering tests
- Instrument the transitions.

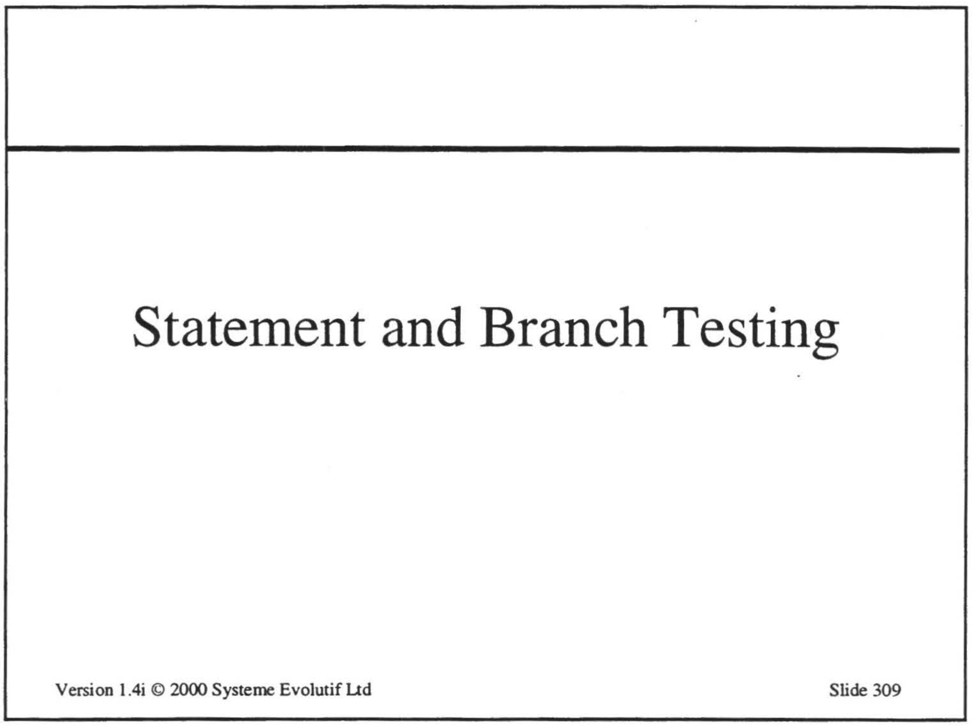
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 307



Early preparation of state transition cases

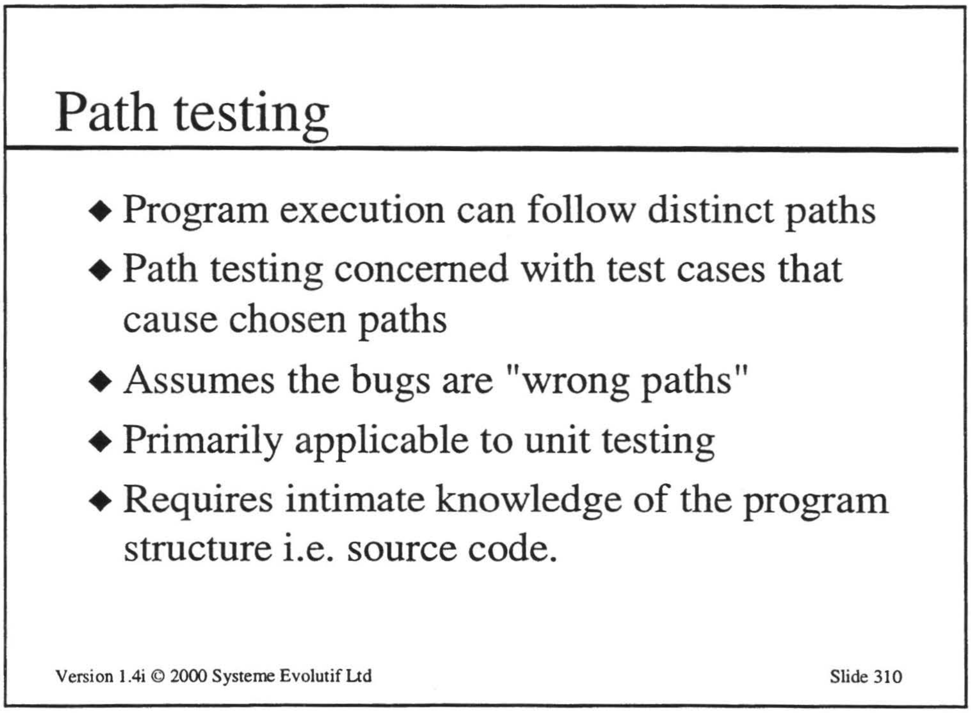
- Particularly useful in early stages
- Have all states been accounted for?
- Have non-equivalent states been merged?
- Have 'impossible' states been excluded
- Dead states?
- Unreachable states?

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 308



Statement and Branch Testing

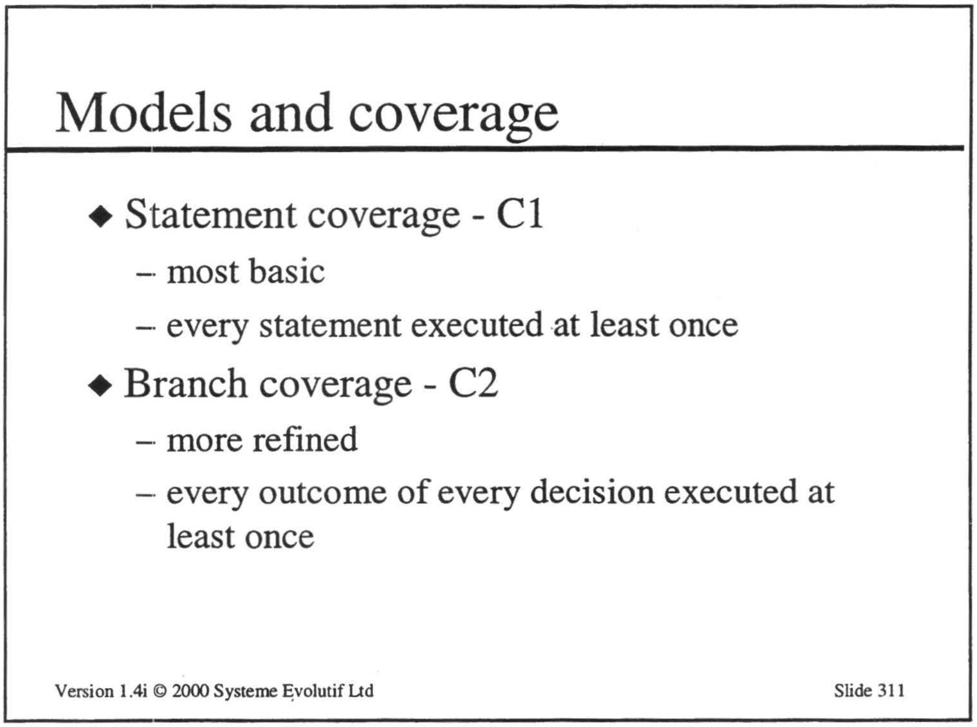
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 309



Path testing

- Program execution can follow distinct paths
- Path testing concerned with test cases that cause chosen paths
- Assumes the bugs are "wrong paths"
- Primarily applicable to unit testing
- Requires intimate knowledge of the program structure i.e. source code.

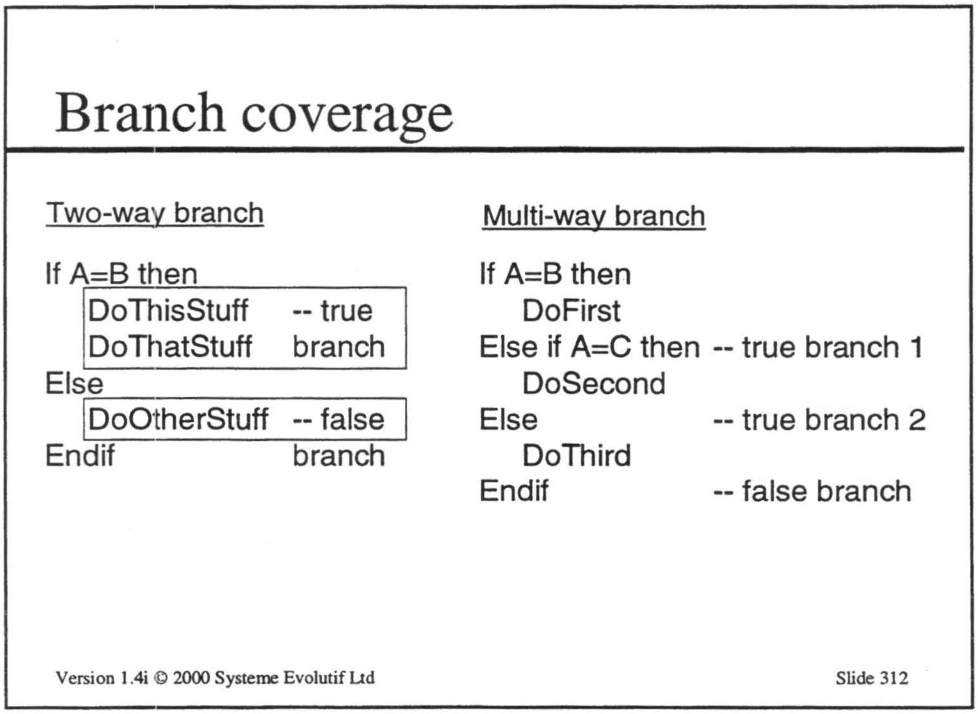
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 310



Models and coverage

- ◆ Statement coverage - C1
  - – most basic
  - – every statement executed at least once
- ◆ Branch coverage - C2
  - – more refined
  - – every outcome of every decision executed at least once

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 311



Branch coverage

Two-way branch

```
If A=B then
    DoThisStuff    -- true
    DoThatStuff    branch
Else
    DoOtherStuff   -- false
Endif              branch
```

Multi-way branch

```
If A=B then
    DoFirst
Else if A=C then  -- true branch 1
    DoSecond
Else              -- true branch 2
    DoThird
Endif             -- false branch
```

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 312

Coverage measurement

- ◆ Achieving coverage is incremental
- ◆ Measured as a percentage

$$\text{Statement Coverage} = \frac{\text{statements\_executed}}{\text{total statements}}$$

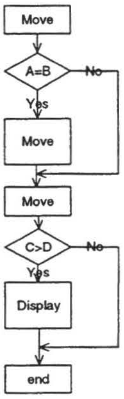
$$\text{Branch Coverage} = \frac{\text{branch\_outcomes\_executed}}{\text{total branch outcomes}}$$

Control flowgraphs

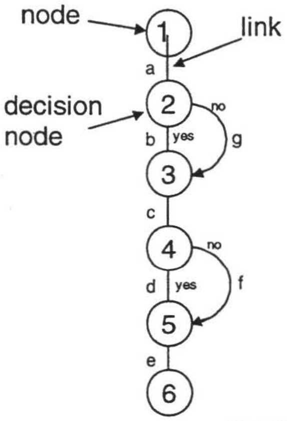
Code

```
MOVE
IF A=B
   MOVE
END-IF
MOVE
IF C>D
   DISPLAY
END-IF
STOP RUN
```

Flow Chart

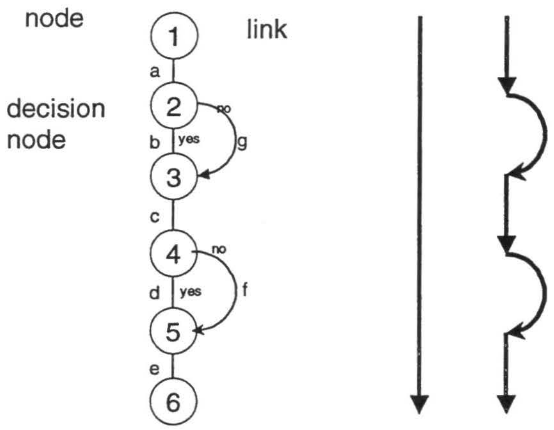


Flow Graph



Sensitising the paths

Flow Graph



- Two test cases are enough to cover all branches
- each path required two conditions to be met

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 315

From paths to test cases

- Need to understand the relationship between paths and test cases
- Process of choosing input values is called "sensitising the path"
- Sometimes difficult for code buried deep in complex programs
- You are free to choose which paths so choose the easiest.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 316

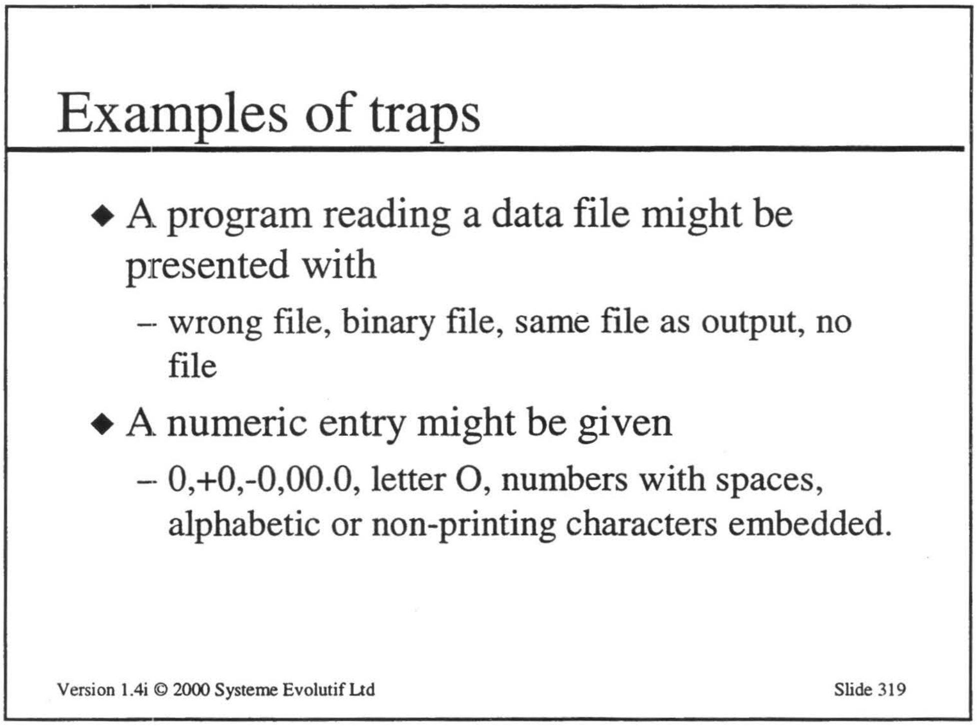
Error-Guessing

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 317

Testing by intuition and experience

- Much ad hoc testing based on intuition and guesswork
- Some people have a knack of finding faults
- Good test cases can be derived in this way
- Should be used as a mopping-up approach.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 318

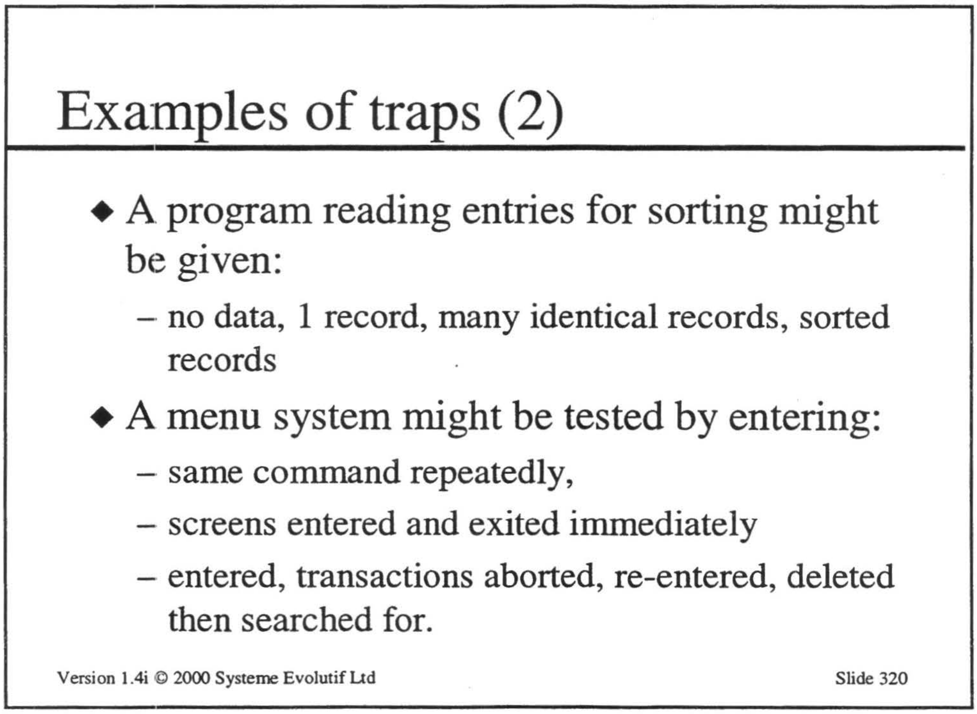


Examples of traps

- A program reading a data file might be presented with
  - wrong file, binary file, same file as output, no file
- A numeric entry might be given
  - 0,+0,-0,00.0, letter O, numbers with spaces, alphabetic or non-printing characters embedded.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 319

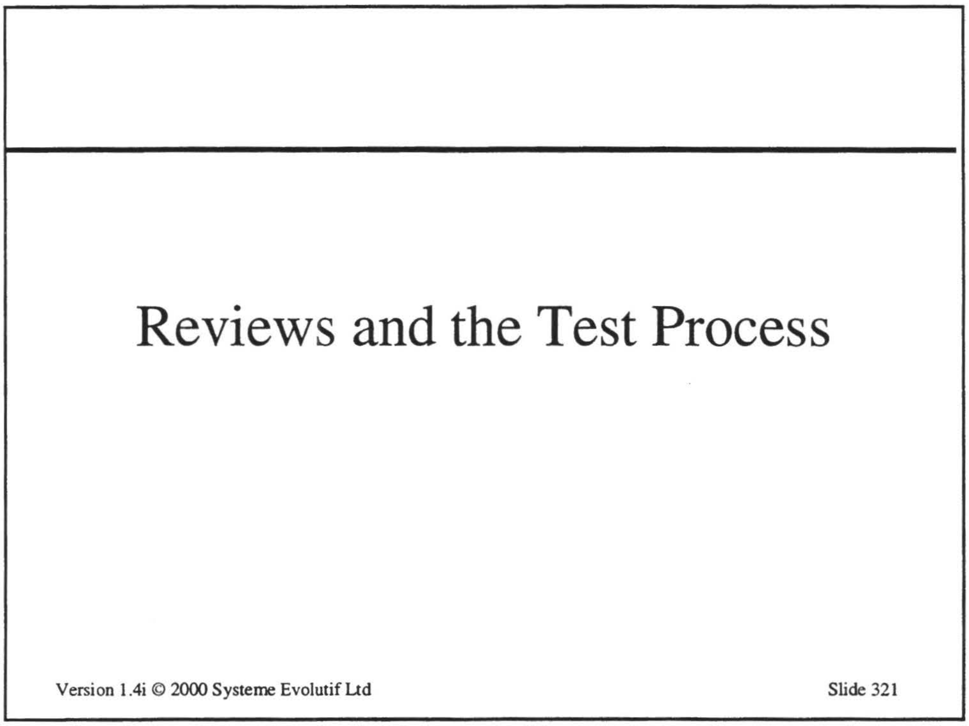


Examples of traps (2)

- A program reading entries for sorting might be given:
  - no data, 1 record, many identical records, sorted records
- A menu system might be tested by entering:
  - same command repeatedly,
  - screens entered and exited immediately
  - entered, transactions aborted, re-entered, deleted then searched for.

Version 1.4i © 2000 Systeme Evolutif Ltd

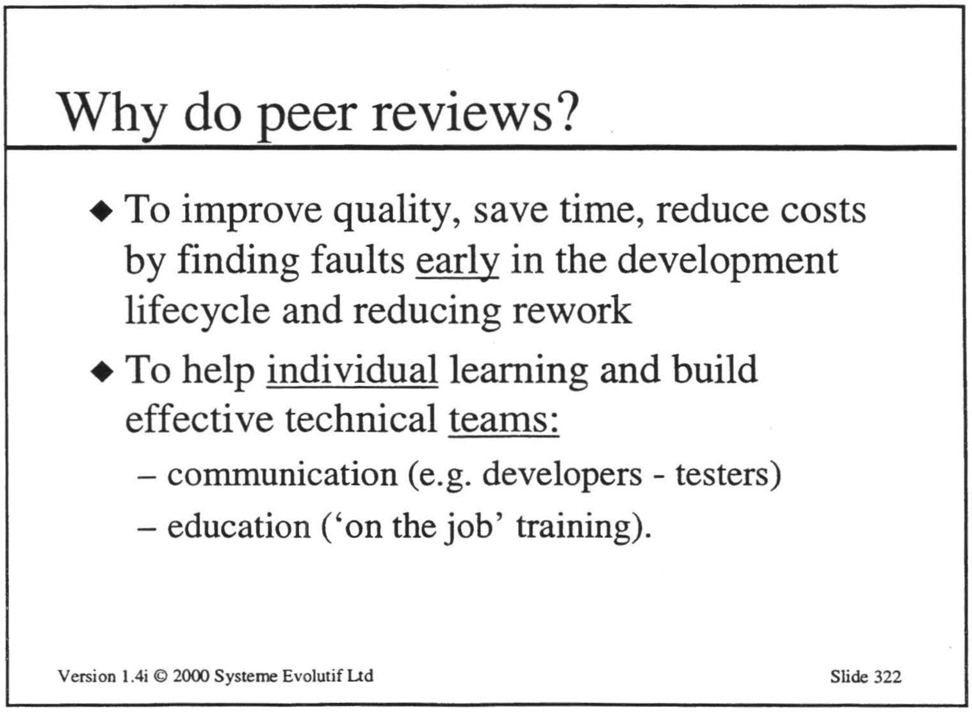
Slide 320



Reviews and the Test Process

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 321

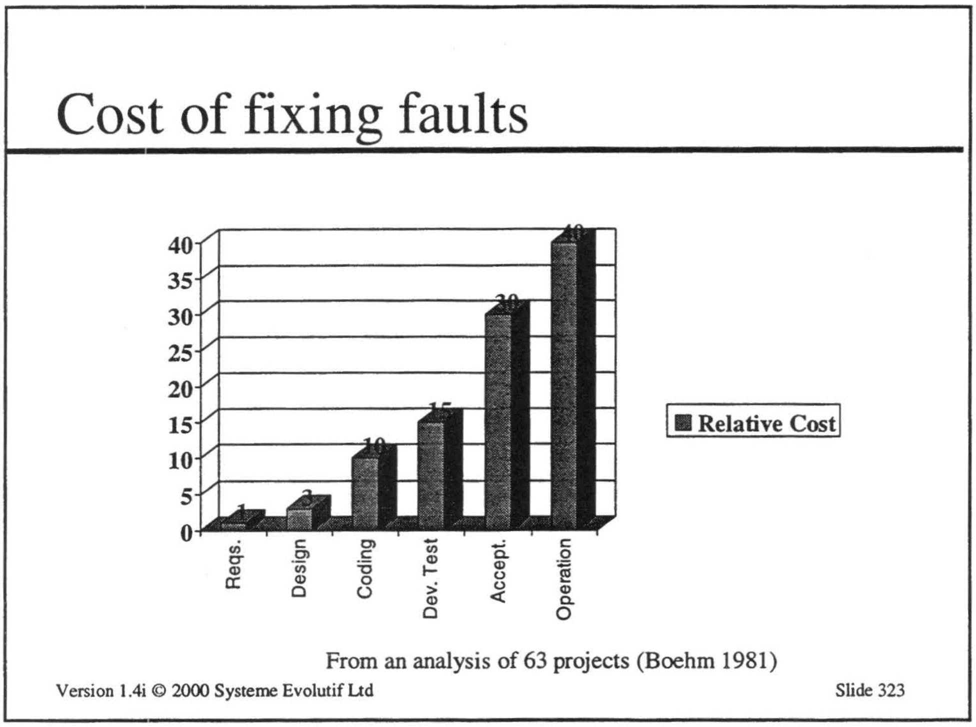


Why do peer reviews?

- To improve quality, save time, reduce costs by finding faults early in the development lifecycle and reducing rework
- To help individual learning and build effective technical teams:
  – communication (e.g. developers - testers)
  – education (‘on the job’ training).

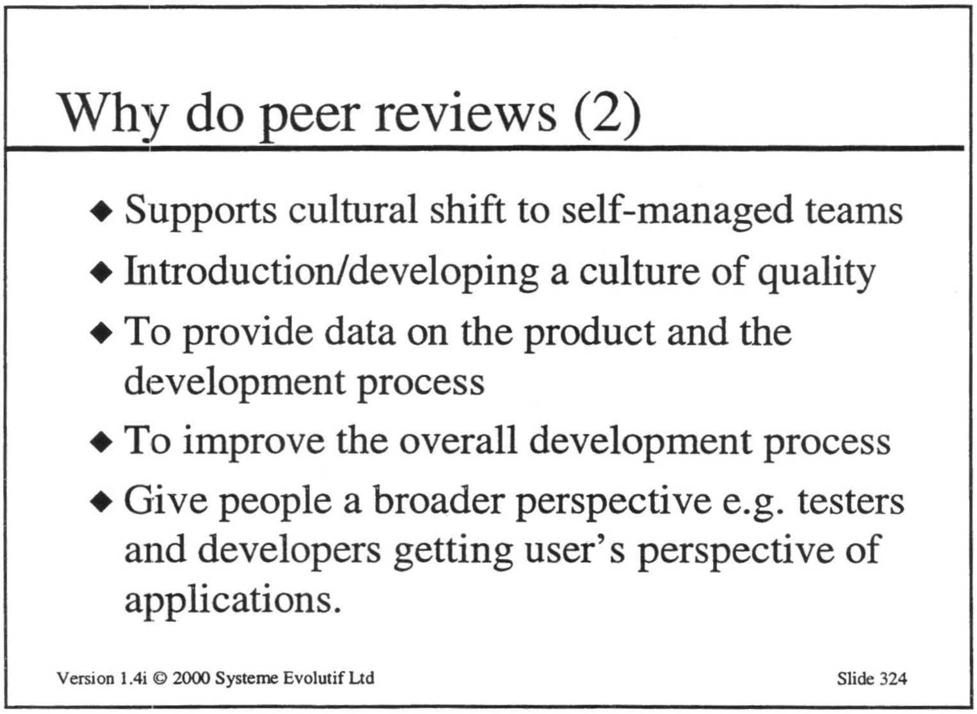
Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 322



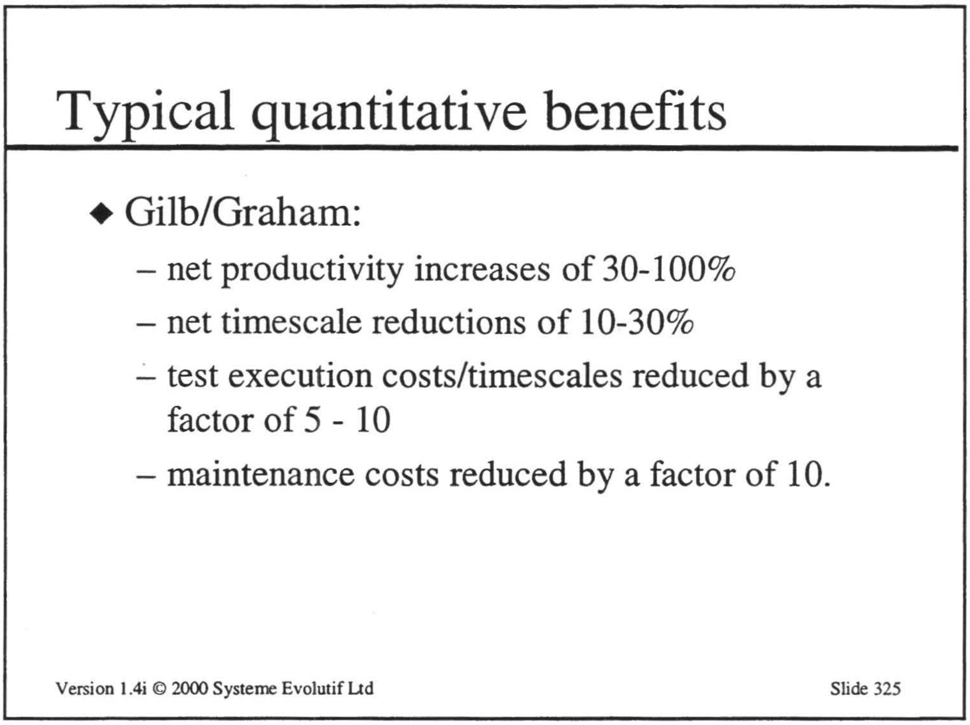
Cost of fixing faults

From an analysis of 63 projects (Boehm 1981)



Why do peer reviews (2)

- ◆ Supports cultural shift to self-managed teams
- ◆ Introduction/developing a culture of quality
- ◆ To provide data on the product and the development process
- ◆ To improve the overall development process
- ◆ Give people a broader perspective e.g. testers and developers getting user's perspective of applications.

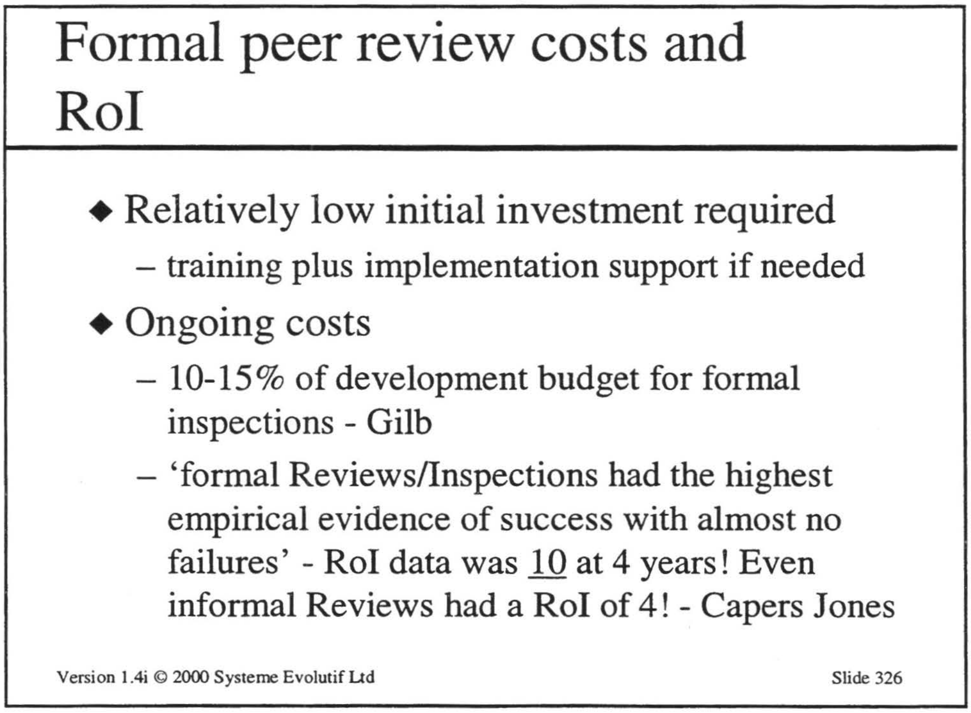


Typical quantitative benefits

- Gilb/Graham:
  - net productivity increases of 30-100%
  - net timescale reductions of 10-30%
  - test execution costs/timescales reduced by a factor of 5 - 10
  - maintenance costs reduced by a factor of 10.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 325



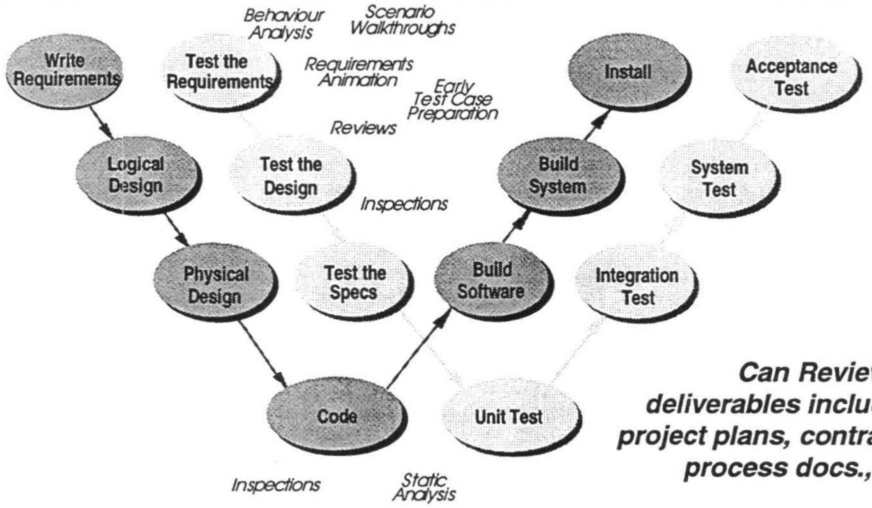
Formal peer review costs and RoI

- Relatively low initial investment required
  - training plus implementation support if needed
- Ongoing costs
  - 10-15% of development budget for formal inspections - Gilb
  - ‘formal Reviews/Inspections had the highest empirical evidence of success with almost no failures’ - RoI data was <u>10</u> at 4 years! Even informal Reviews had a RoI of 4! - Capers Jones

Version 1.4i © 2000 Systeme Evolutif Ltd

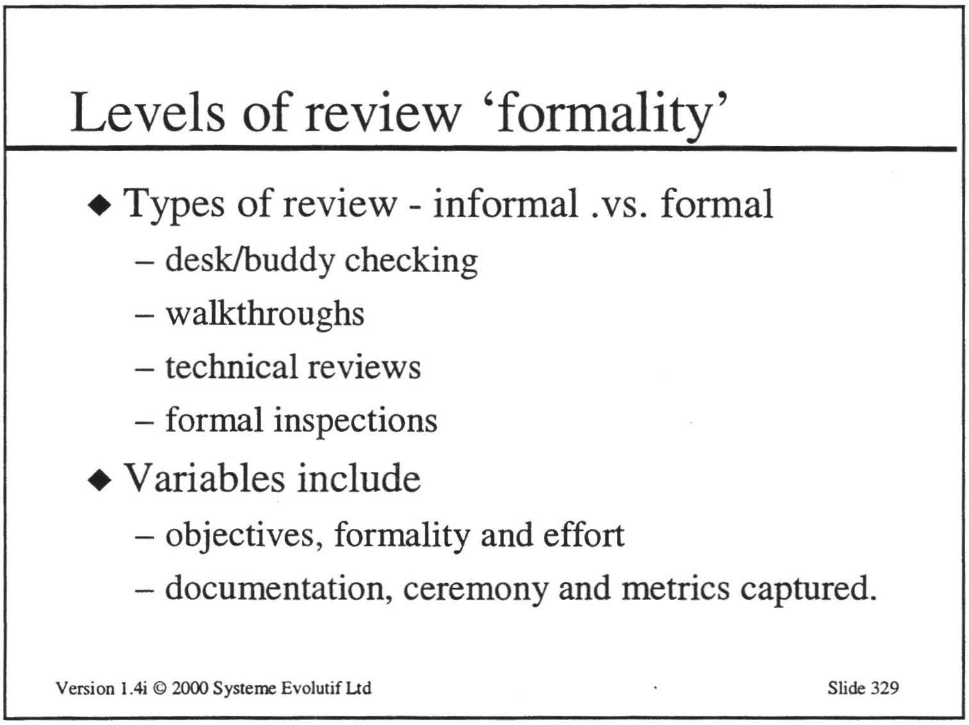
Slide 326

What and when to review



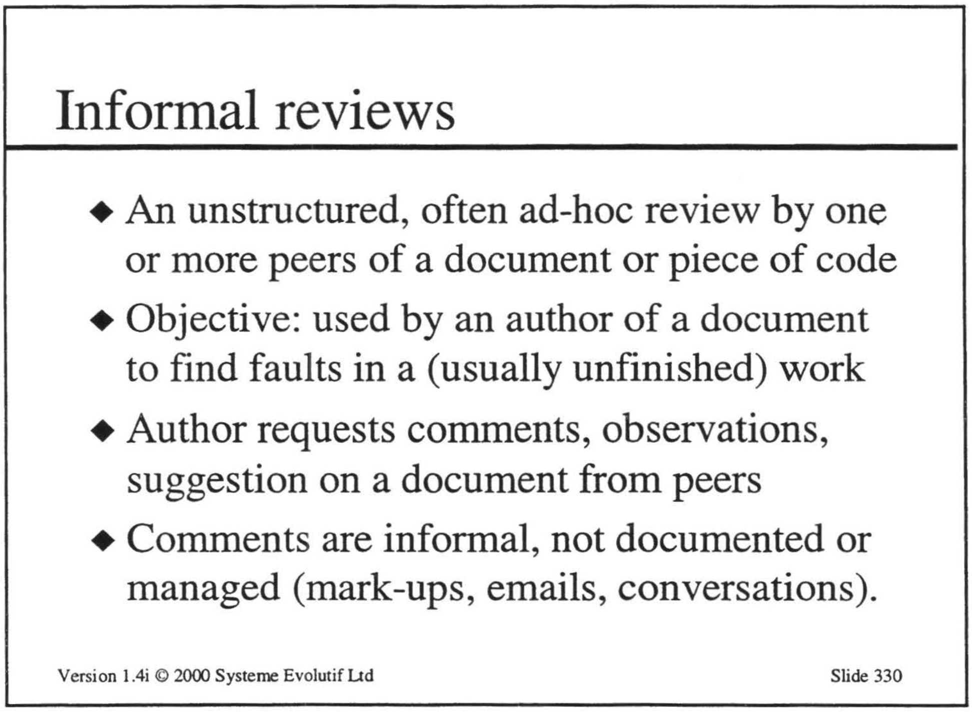
Can Review all deliverables including project plans, contracts, process docs., etc.

Types of Review



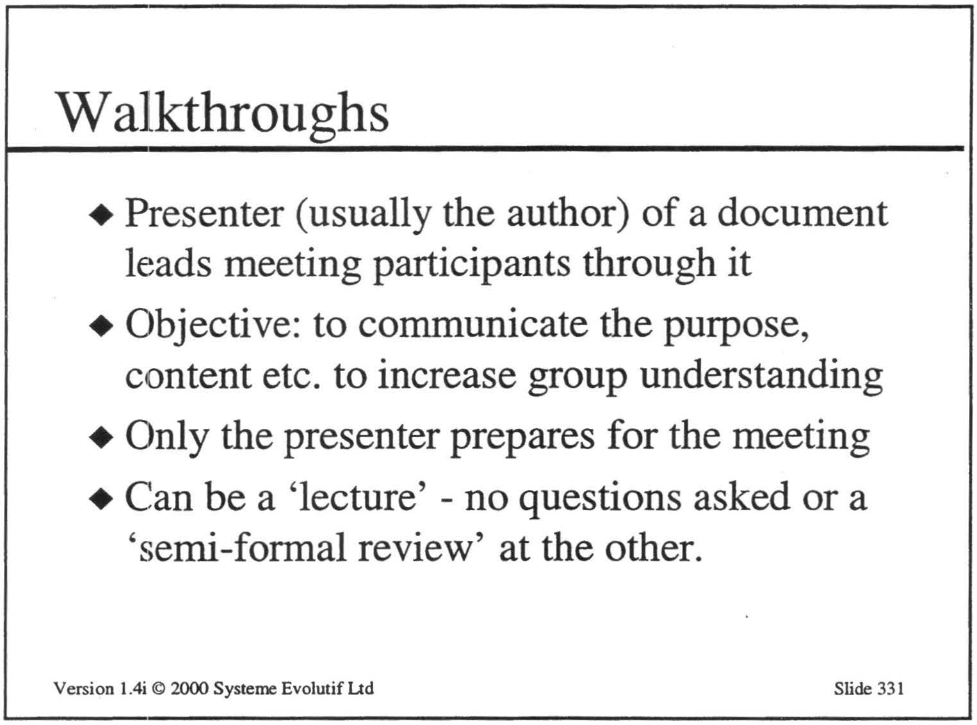
Levels of review 'formality'

- Types of review - informal .vs. formal
  - desk/buddy checking
  - walkthroughs
  - technical reviews
  - formal inspections
- Variables include
  - objectives, formality and effort
  - documentation, ceremony and metrics captured.



Informal reviews

- An unstructured, often ad-hoc review by one or more peers of a document or piece of code
- Objective: used by an author of a document to find faults in a (usually unfinished) work
- Author requests comments, observations, suggestion on a document from peers
- Comments are informal, not documented or managed (mark-ups, emails, conversations).

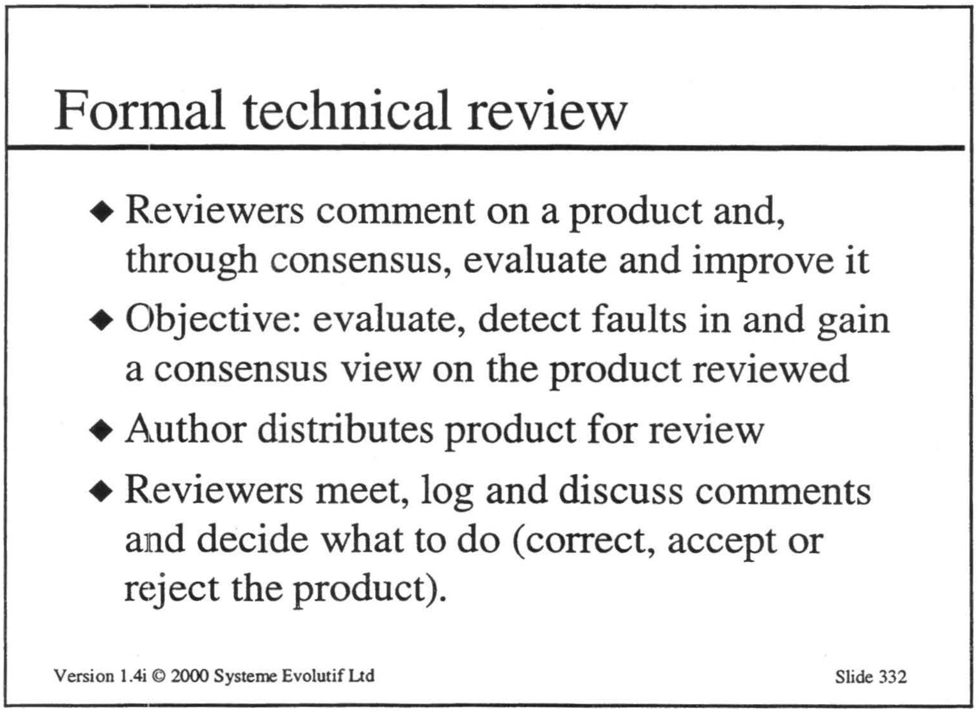


Walkthroughs

- Presenter (usually the author) of a document leads meeting participants through it
- Objective: to communicate the purpose, content etc. to increase group understanding
- Only the presenter prepares for the meeting
- Can be a ‘lecture’ - no questions asked or a ‘semi-formal review’ at the other.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 331



Formal technical review

- Reviewers comment on a product and, through consensus, evaluate and improve it
- Objective: evaluate, detect faults in and gain a consensus view on the product reviewed
- Author distributes product for review
- Reviewers meet, log and discuss comments and decide what to do (correct, accept or reject the product).

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 332

Inspections

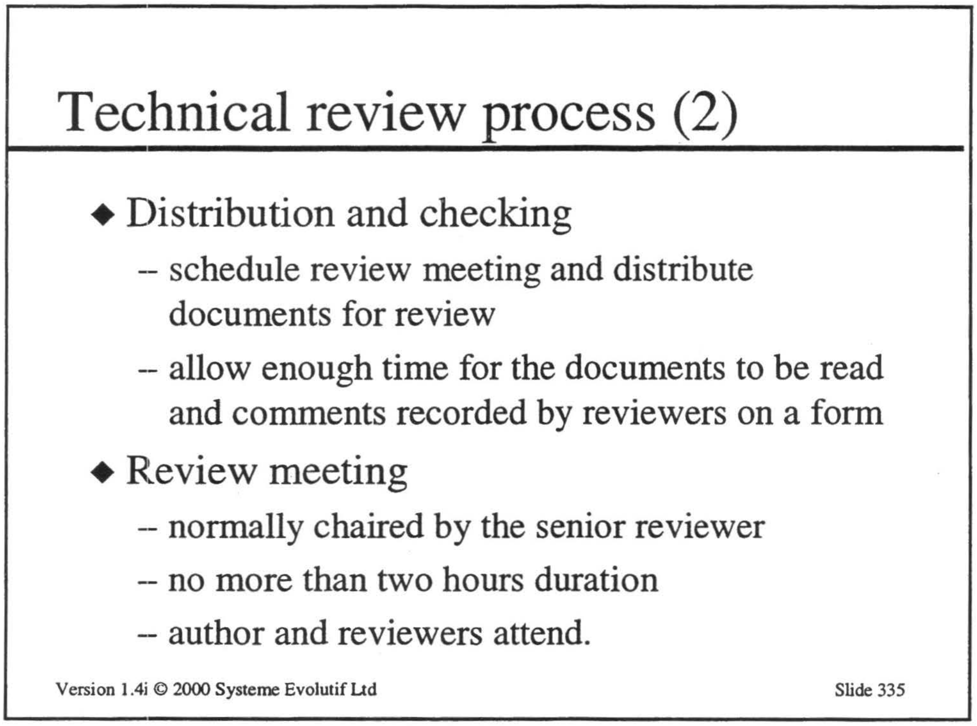
- A formal process, roles and documentation used to detect faults in a product
- Objective: to detect faults and improve the qualify of a product
- Inspections are planned, with a moderator managing the process; inspectors check the product against rules, standards baselines
- Logging meetings document faults and these are fixed by the author.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 333

Technical review process (1)

- Planning - allocate schedule time, resources
- Select review team
  - 3/4 technical staff who can raise informed criticism, one of whom is more senior and can make technical decisions
  - some invited to comment on sub-sets of the material that affect their work
  - others invited to submit written comments, but not attend the review meeting.

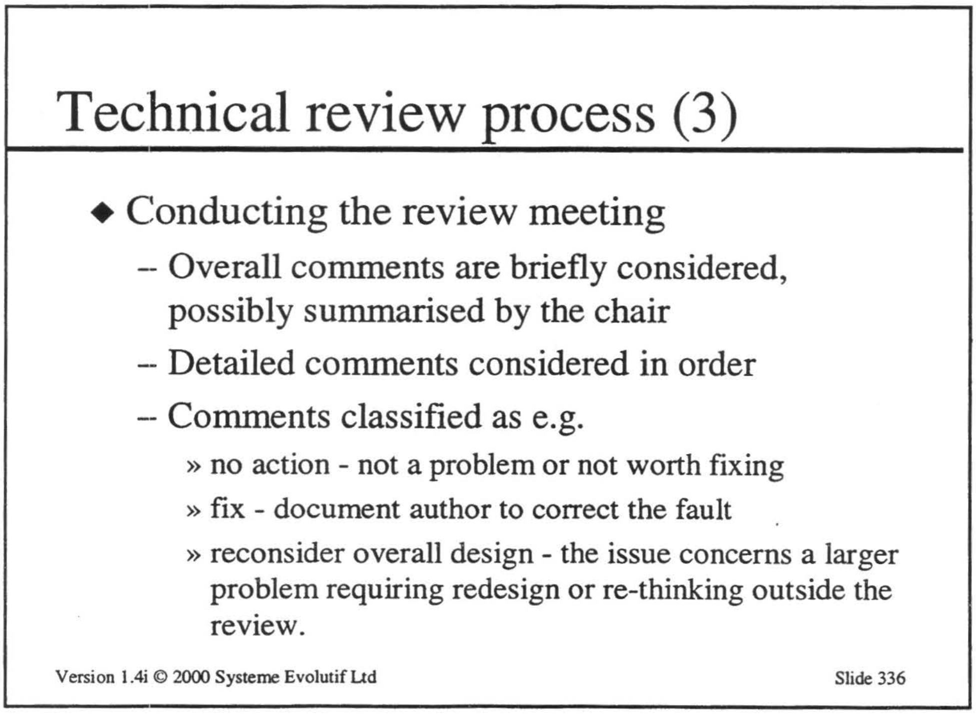
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 334



Technical review process (2)

- Distribution and checking
  - schedule review meeting and distribute documents for review
  - allow enough time for the documents to be read and comments recorded by reviewers on a form
- Review meeting
  - normally chaired by the senior reviewer
  - no more than two hours duration
  - author and reviewers attend.

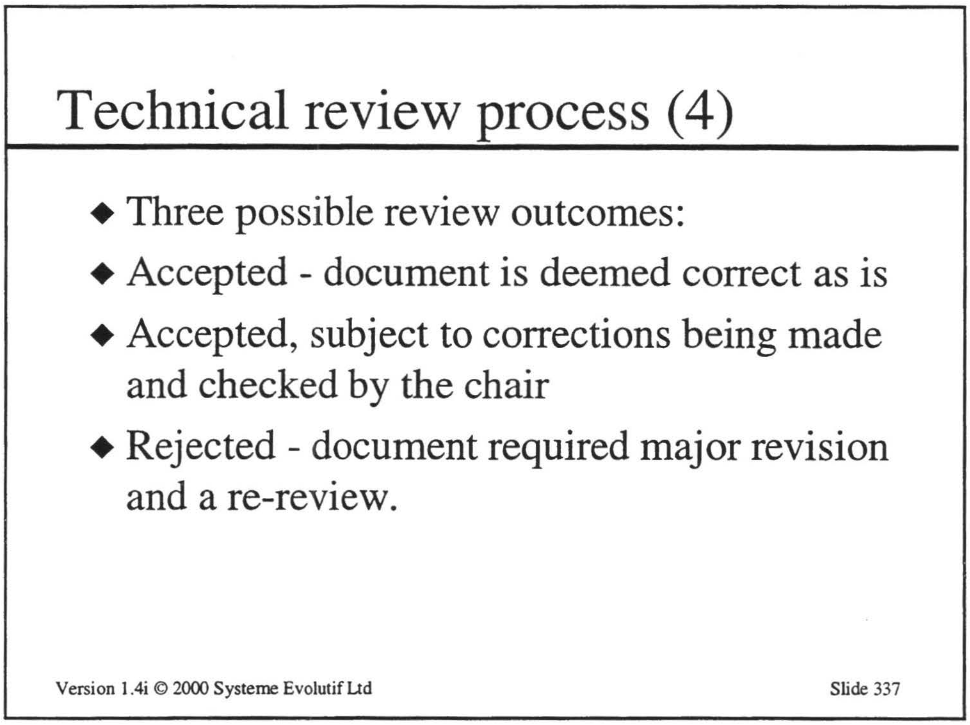
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 335



Technical review process (3)

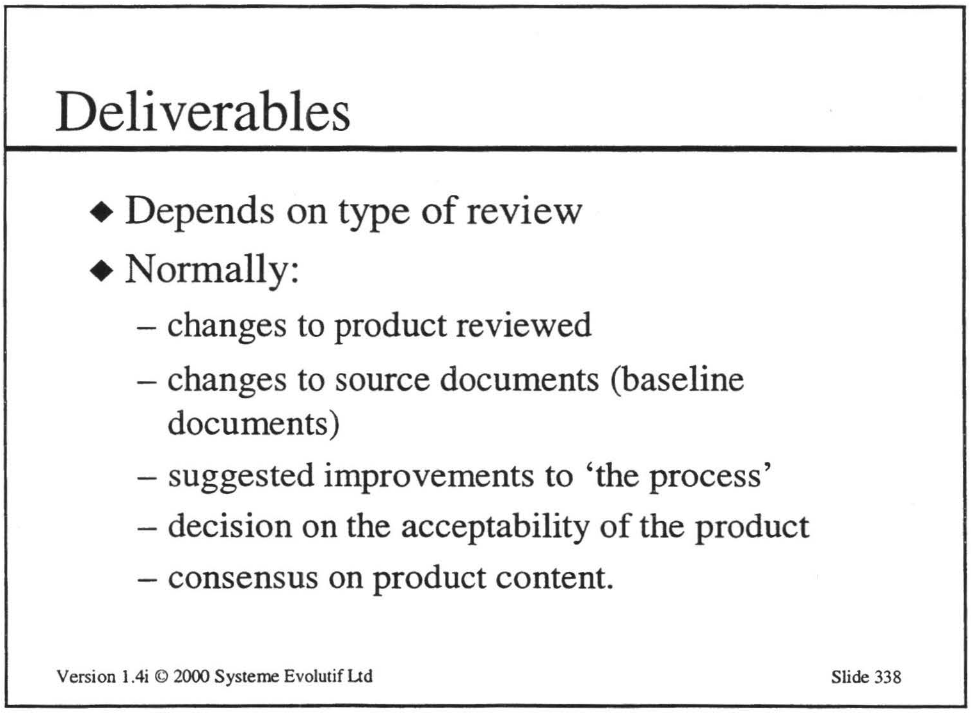
- Conducting the review meeting
  - Overall comments are briefly considered, possibly summarised by the chair
  - Detailed comments considered in order
  - Comments classified as e.g.
    - » no action - not a problem or not worth fixing
    - » fix - document author to correct the fault
    - » reconsider overall design - the issue concerns a larger problem requiring redesign or re-thinking outside the review.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 336



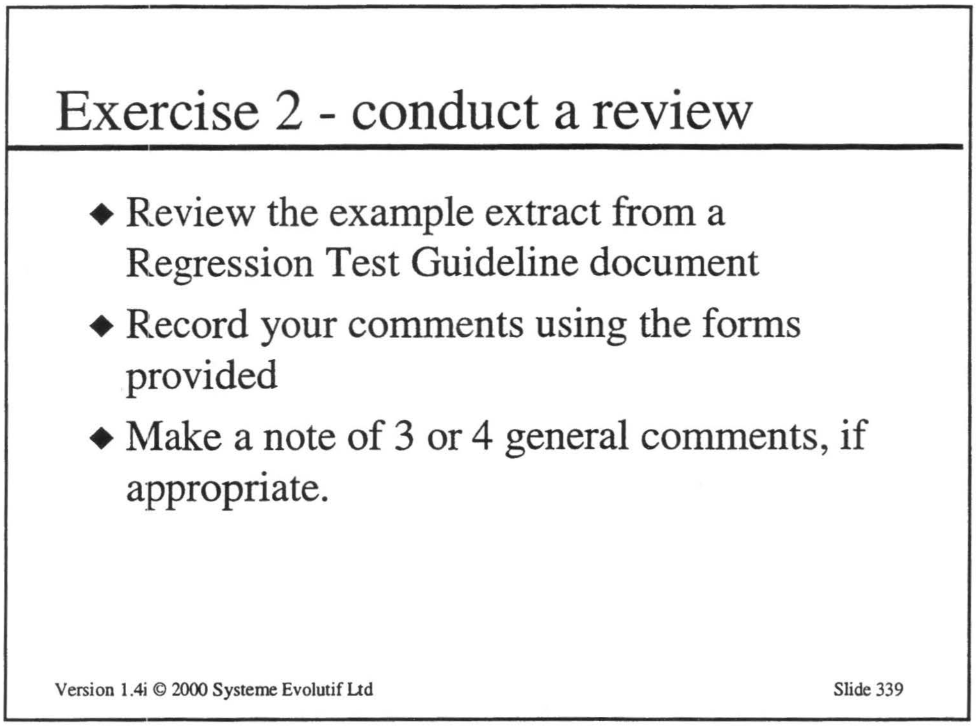
Technical review process (4)

- Three possible review outcomes:
- Accepted - document is deemed correct as is
- Accepted, subject to corrections being made and checked by the chair
- Rejected - document required major revision and a re-review.



Deliverables

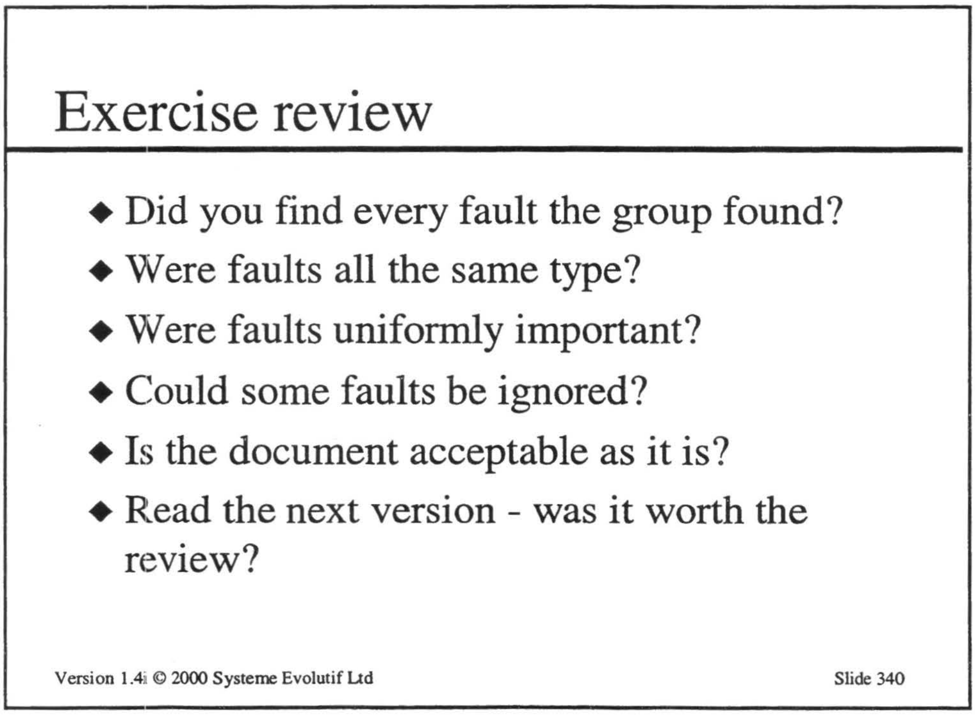
- Depends on type of review
- Normally:
  - changes to product reviewed
  - changes to source documents (baseline documents)
  - suggested improvements to ‘the process’
  - decision on the acceptability of the product
  - consensus on product content.



Exercise 2 - conduct a review

- Review the example extract from a Regression Test Guideline document
- Record your comments using the forms provided
- Make a note of 3 or 4 general comments, if appropriate.

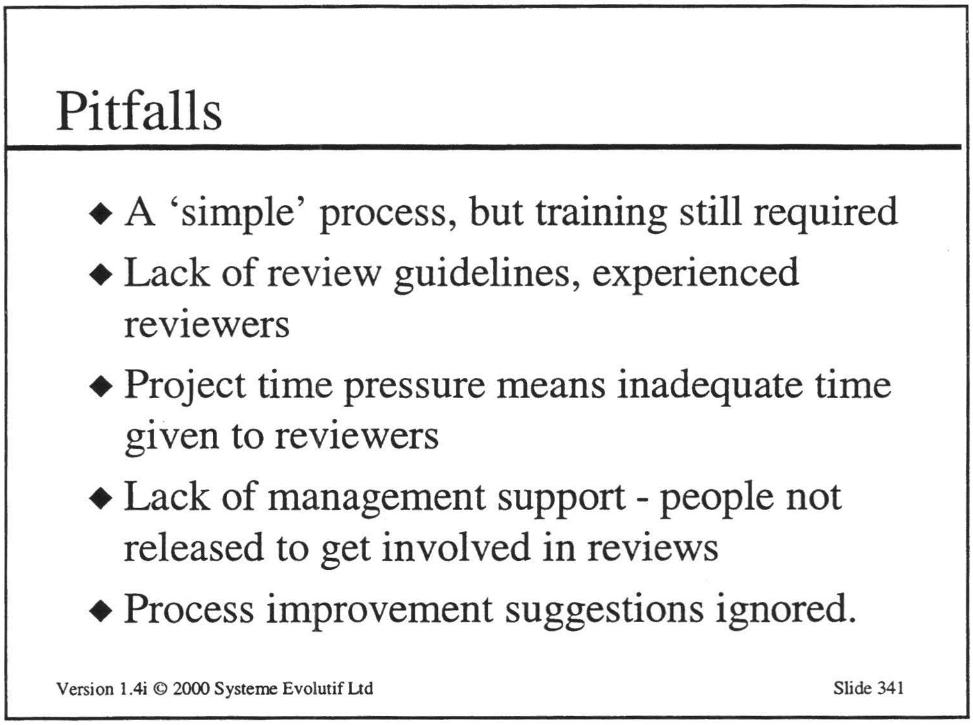
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 339



Exercise review

- Did you find every fault the group found?
- Were faults all the same type?
- Were faults uniformly important?
- Could some faults be ignored?
- Is the document acceptable as it is?
- Read the next version - was it worth the review?

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 340

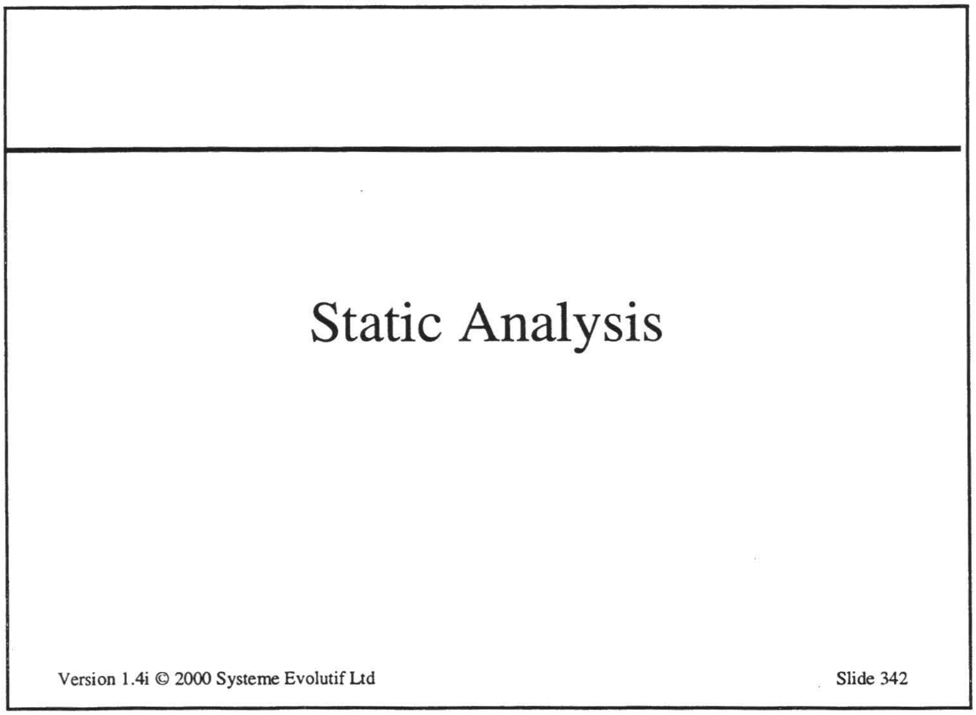


Pitfalls

- A 'simple' process, but training still required
- Lack of review guidelines, experienced reviewers
- Project time pressure means inadequate time given to reviewers
- Lack of management support - people not released to get involved in reviews
- Process improvement suggestions ignored.

Version 1.4i © 2000 Systeme Evolutif Ltd

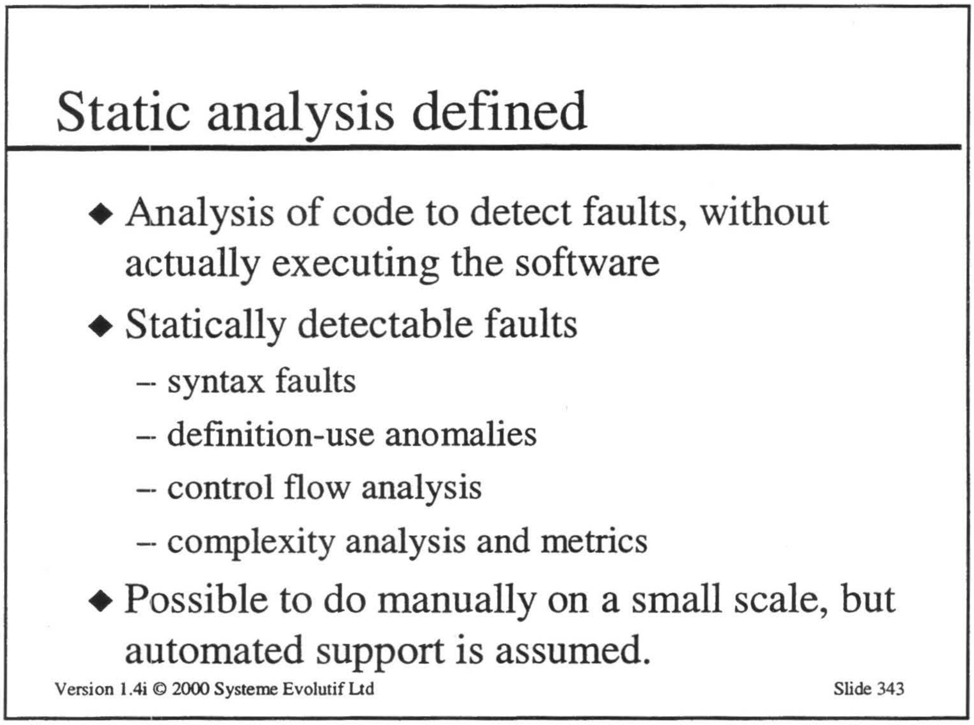
Slide 341



Static Analysis

Version 1.4i © 2000 Systeme Evolutif Ltd

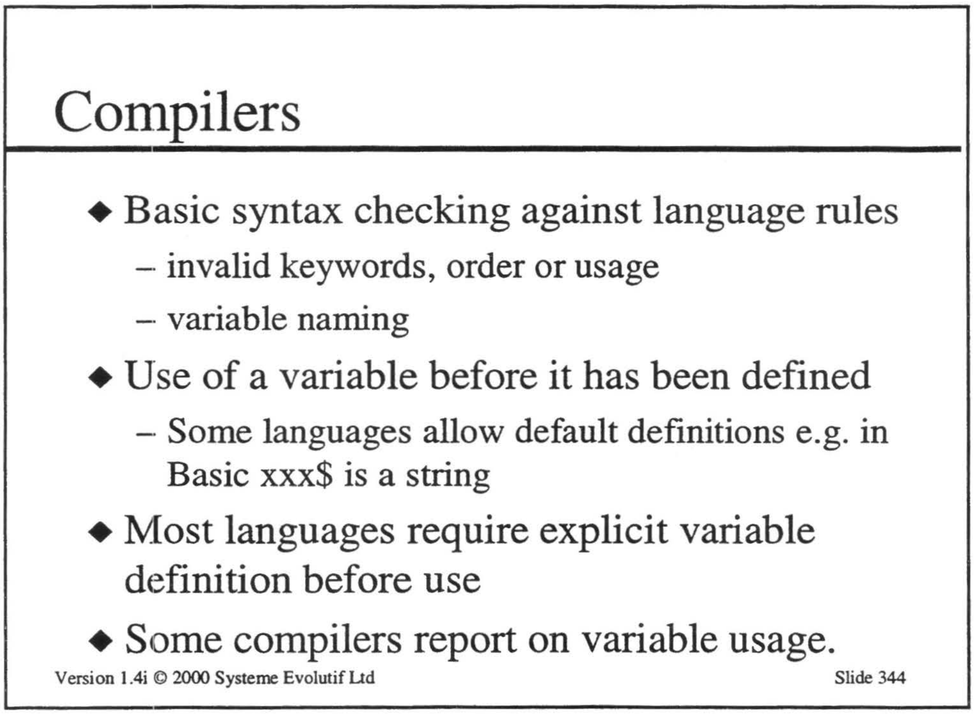
Slide 342



Static analysis defined

- Analysis of code to detect faults, without actually executing the software
- Statically detectable faults
  - syntax faults
  - definition-use anomalies
  - control flow analysis
  - complexity analysis and metrics
- Possible to do manually on a small scale, but automated support is assumed.

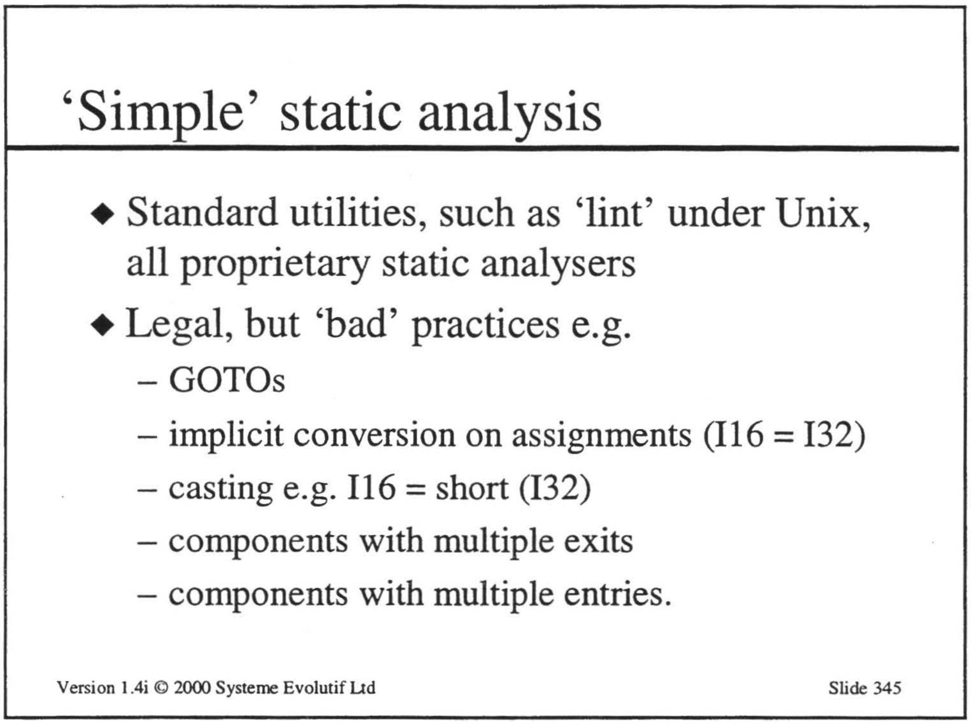
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 343



Compilers

- Basic syntax checking against language rules
  - invalid keywords, order or usage
  - variable naming
- Use of a variable before it has been defined
  - Some languages allow default definitions e.g. in Basic xxx$ is a string
- Most languages require explicit variable definition before use
- Some compilers report on variable usage.

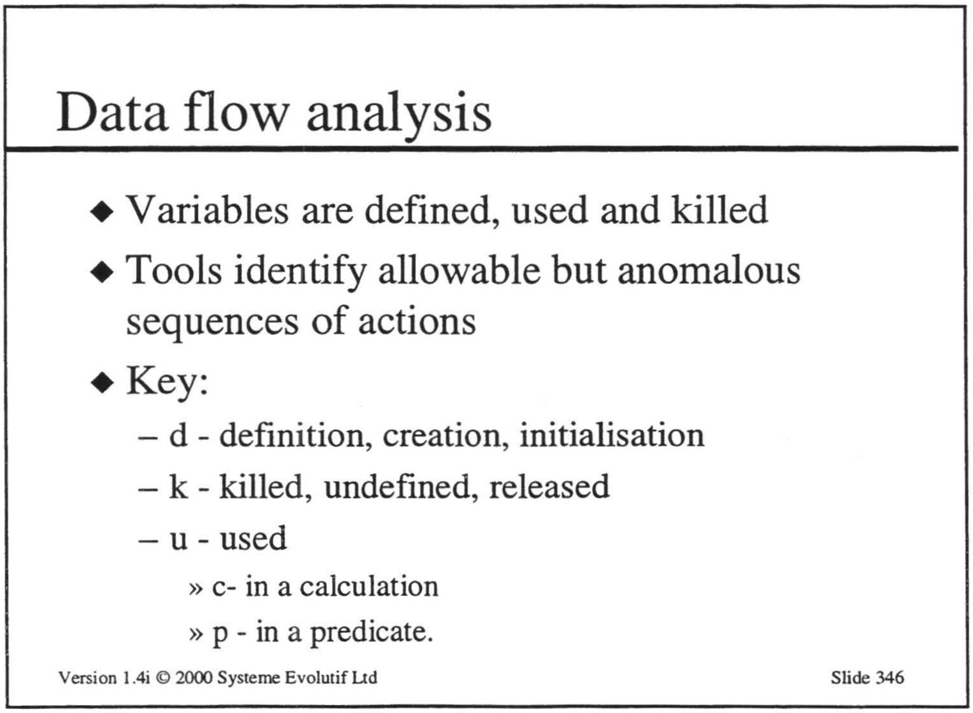
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 344



'Simple' static analysis

- ◆ Standard utilities, such as 'lint' under Unix, all proprietary static analysers
- ◆ Legal, but 'bad' practices e.g.
  - – GOTOs
  - – implicit conversion on assignments (I16 = I32)
  - – casting e.g. I16 = short (I32)
  - – components with multiple exits
  - – components with multiple entries.

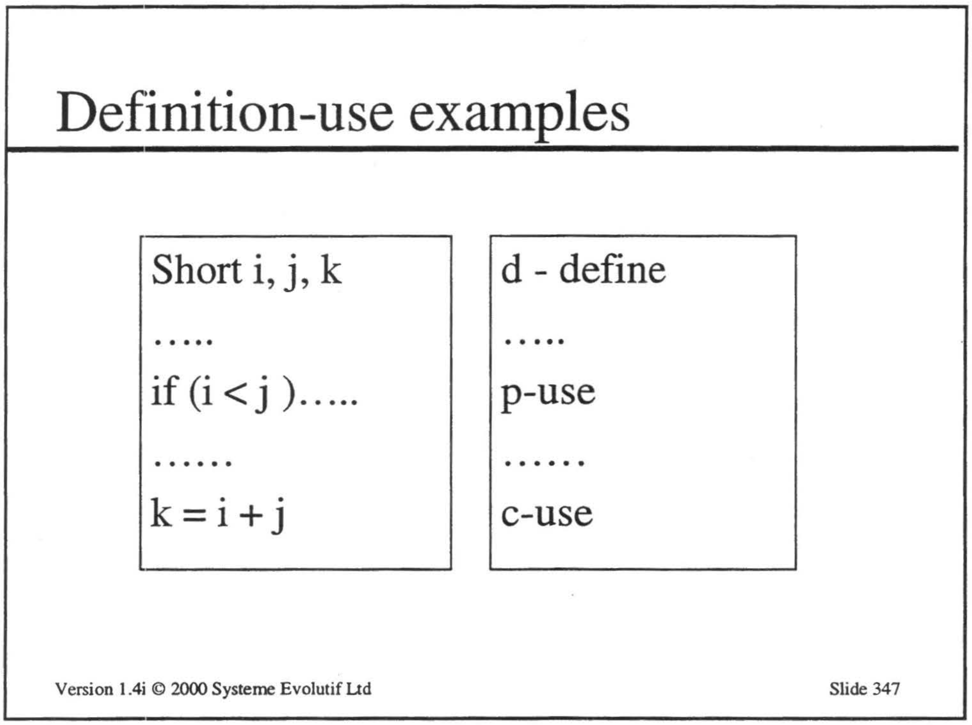
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 345



Data flow analysis

- ◆ Variables are defined, used and killed
- ◆ Tools identify allowable but anomalous sequences of actions
- ◆ Key:
  - – d - definition, creation, initialisation
  - – k - killed, undefined, released
  - – u - used
    - » c- in a calculation
    - » p - in a predicate.

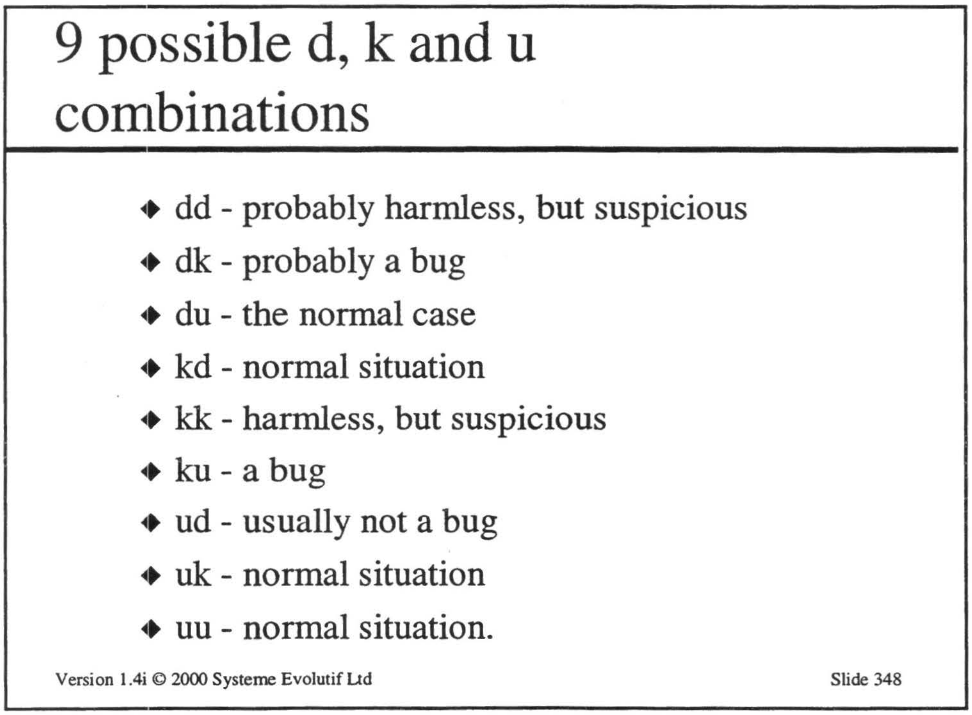
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 346



Definition-use examples

Short i, j, k	d - define
…..	…..
if (i < j)…..	p-use
……	……
k = i + j	c-use

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 347



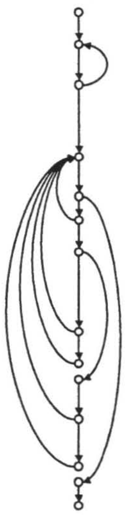
9 possible d, k and u combinations

- dd - probably harmless, but suspicious
- dk - probably a bug
- du - the normal case
- kd - normal situation
- kk - harmless, but suspicious
- ku - a bug
- ud - usually not a bug
- uk - normal situation
- uu - normal situation.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 348

Code and control-flow graph

```
START-OF-RUN.
       MOVE SPACES TO MATCH-CONTROL MATCH-INPUT-ADDRESS.
       MOVE 'N' TO MC-REBATE-IND.
       MOVE 1 TO PTR-1.
A004.
       MOVE 0 TO MC-ROUTINE-FLAG (PTR-1).
       ADD 1 TO PTR-1.
       IF PTR-1 NOT > 36
                      GO TO A004.
       MOVE SPACES TO FMFILE.
       MOVE 2 TO PGZ-FUNCTION.
                      DISPLAY 'Demonstration1987'
                      LINE 1 COLUMN 19 ERASE SCREEN.
       DISPLAY 'PROBE Enquiry Facility    - '
                                          LINE 1 COLUMN 8 ERASE SCREEN.
A020.
       MOVE SPACES TO MATCH-INPUT-ADDRESS MATCH-OUTPUT-ADDRESS.
       MOVE ZERO TO MIA-NO-LINES.
                      ACCEPT MIA-ADD-LINE (1)  LINE 3 COLUMN 19.
       IF MIA-ADD-LINE (1) = 'END'
           GO TO A900-END-OF-RUN.
       IF MIA-ADD-LINE (1) = 'ZZZZ'
           GO TO A020.
       ACCEPT MIA-ADD-LINE (2)  LINE 4 COLUMN 19.
                      IF MIA-ADD-LINE (1) = SPACES AND
                              MIA-ADD-LINE (2) = SPACES
           GO TO A100.
       MOVE 0 TO TWINKLE-COUNT.
       INSPECT MIA-ADD-LINE (2) TALLYING TWINKLE-COUNT
                                          FOR ALL '*'.
       INSPECT MIA-ADD-LINE (2) TALLYING TWINKLE-COUNT
                                          FOR ALL '?'.
       IF TWINKLE-COUNT NOT = 0
                      DISPLAY SPACE LINE 8 COLUMN 1 ERASE END SCREEN
                           GO TO A020.
       DISPLAY SPACE LINE 8 COLUMN 1 ERASE END SCREEN.
       MOVE 5 TO MIA-NO-LINES.
       GO TO A020.
A100.
       DISPLAY 'Postcode:' LINE 8 COLUMN 5 ERASE LINE.
       ACCEPT DENORM-LINE LINE 8 COLUMN 19.
       DISPLAY SPACE LINE 8 COLUMN 1 ERASE END SCREEN.
       IF DENORM-LINE = SPACES
                           GO TO A020.
       MOVE SPACES TO PC-PREM-ID.
       UNSTRING DENORM-LINE DELIMITED '/'
           INTO PCI-POSTCODE PC-PREM-ID.
       MOVE 0 TO PC-PREM-DISP.
       GO TO A020.
A900-END-OF-RUN.
       DISPLAY ' ' LINE 24.
       MOVE 3 TO PGZ-FUNCTION.
                      STOP RUN.
```



Version 1.4i © 2000 Systeme Evolutif Ltd

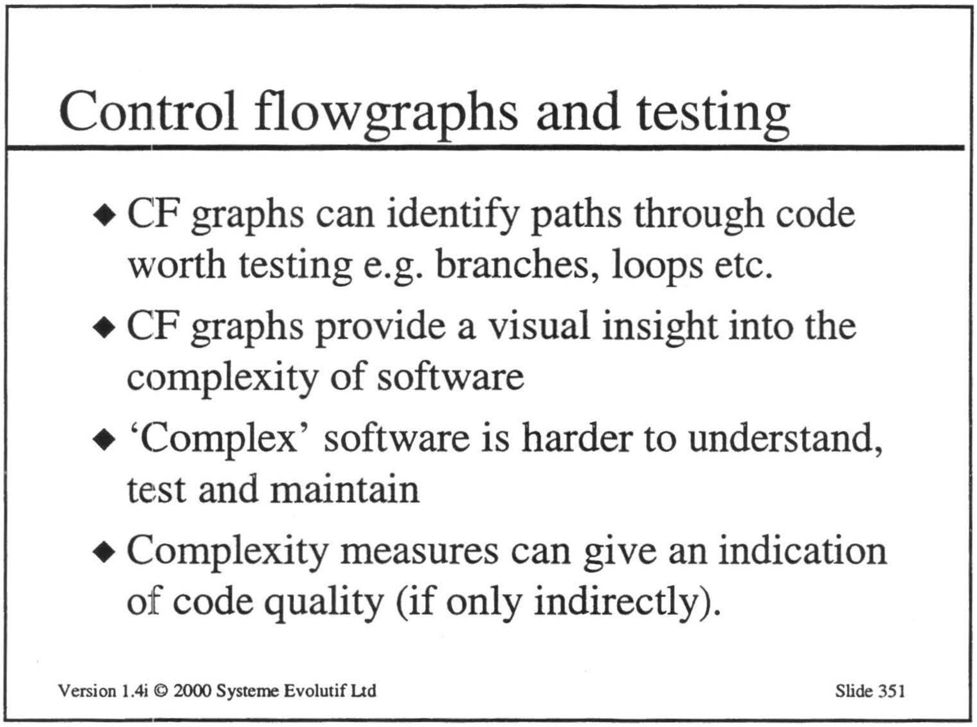
Slide 349

Control flow graph

- ◆ A graphical model of code consisting of nodes and links
- ◆ A link represents a sequence of executable statements, no exit (decisions) or entries (labels)
- ◆ Node represents
  - – a decision with 2 or more outcomes
  - – a statement to which code can jump to.

Version 1.4i © 2000 Systeme Evolutif Ltd

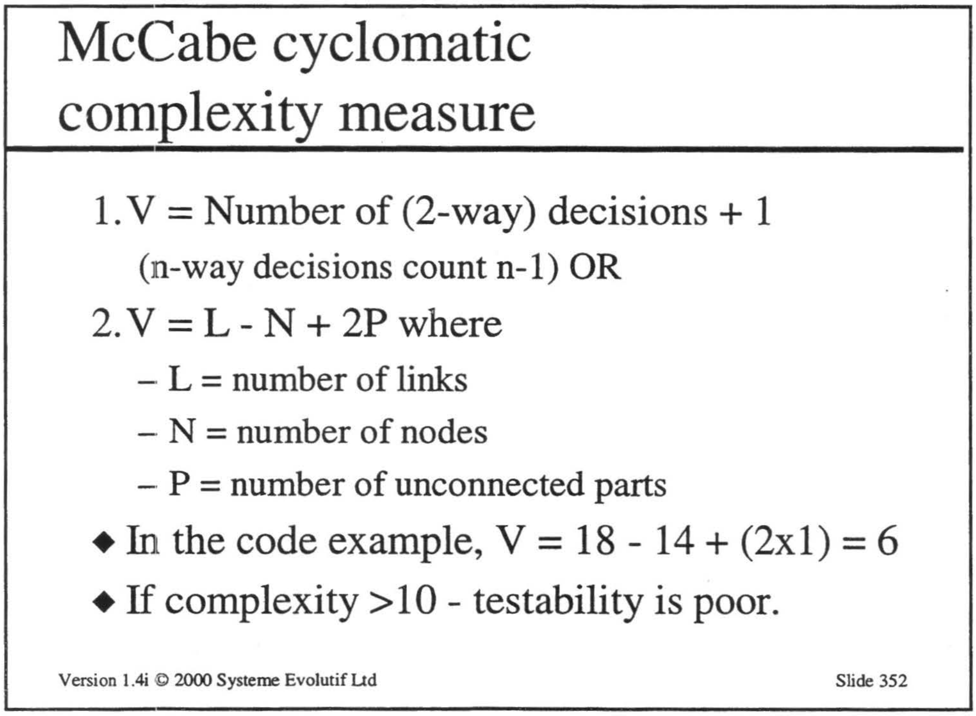
Slide 350



Control flowgraphs and testing

- CF graphs can identify paths through code worth testing e.g. branches, loops etc.
- CF graphs provide a visual insight into the complexity of software
- 'Complex' software is harder to understand, test and maintain
- Complexity measures can give an indication of code quality (if only indirectly).

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 351

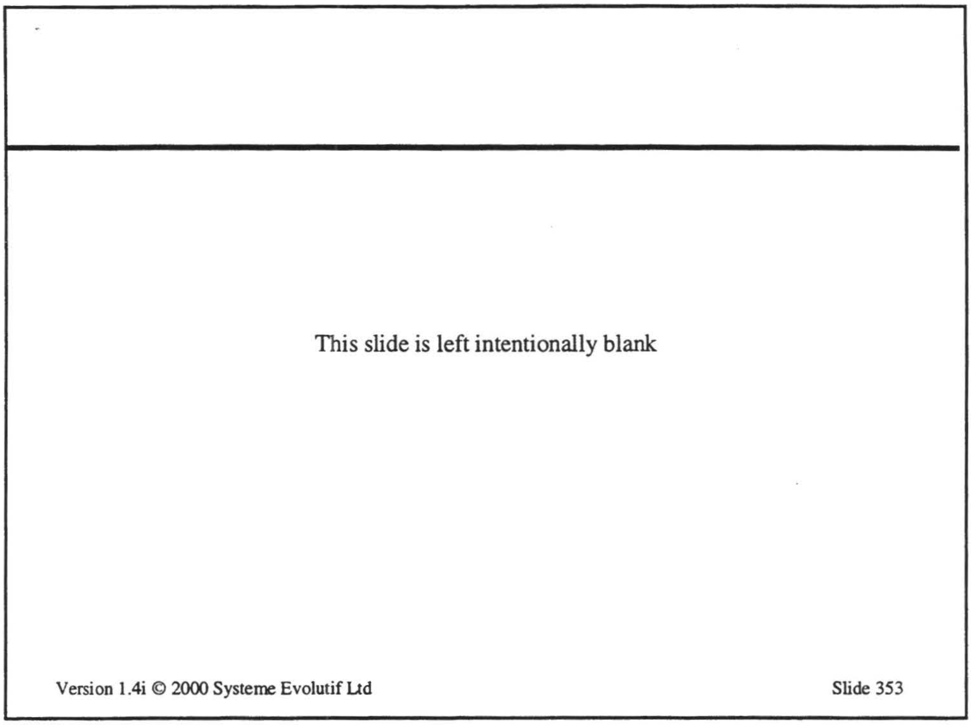


McCabe cyclomatic complexity measure

1. V = Number of (2-way) decisions + 1
   (n-way decisions count n-1) OR
2. V = L - N + 2P where
   - L = number of links
   - N = number of nodes
   - P = number of unconnected parts

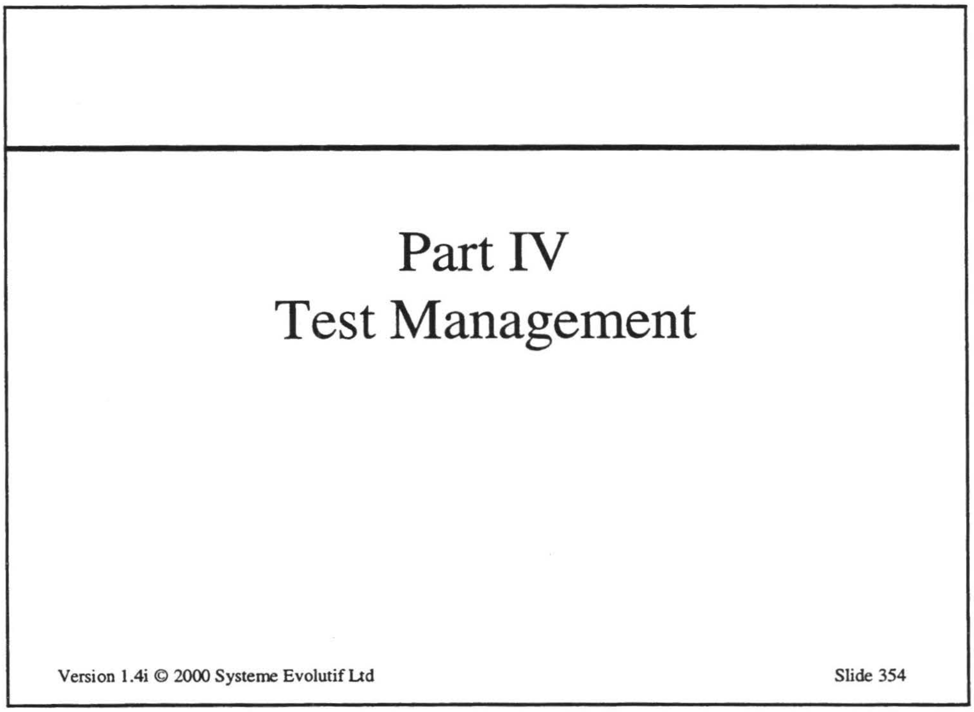
- In the code example, V = 18 - 14 + (2x1) = 6
- If complexity >10 - testability is poor.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 352


This slide is left intentionally blank

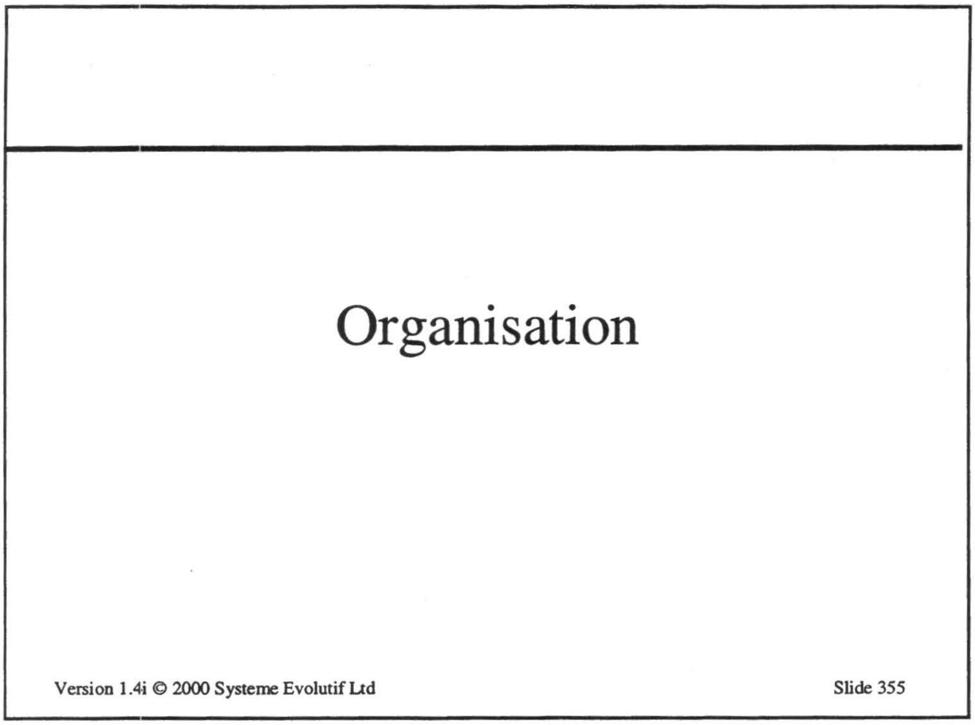
Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 353


Part IV
Test Management

Version 1.4i © 2000 Systeme Evolutif Ltd

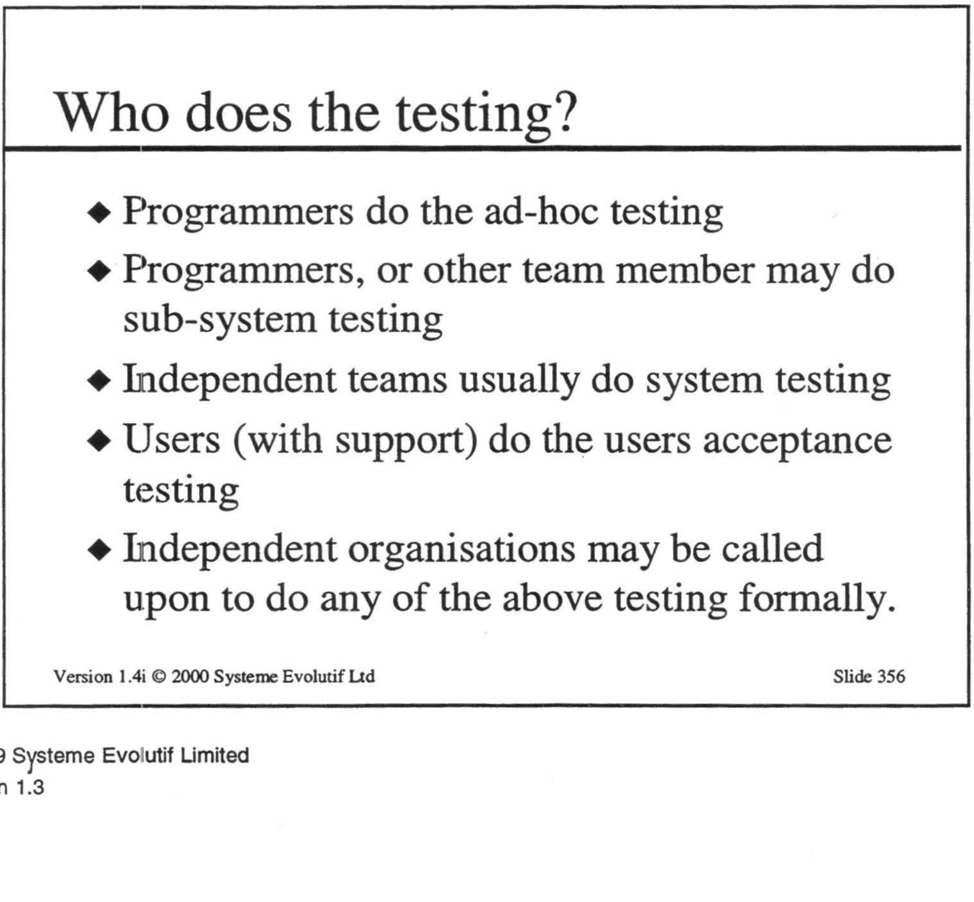
Slide 354



Organisation

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 355

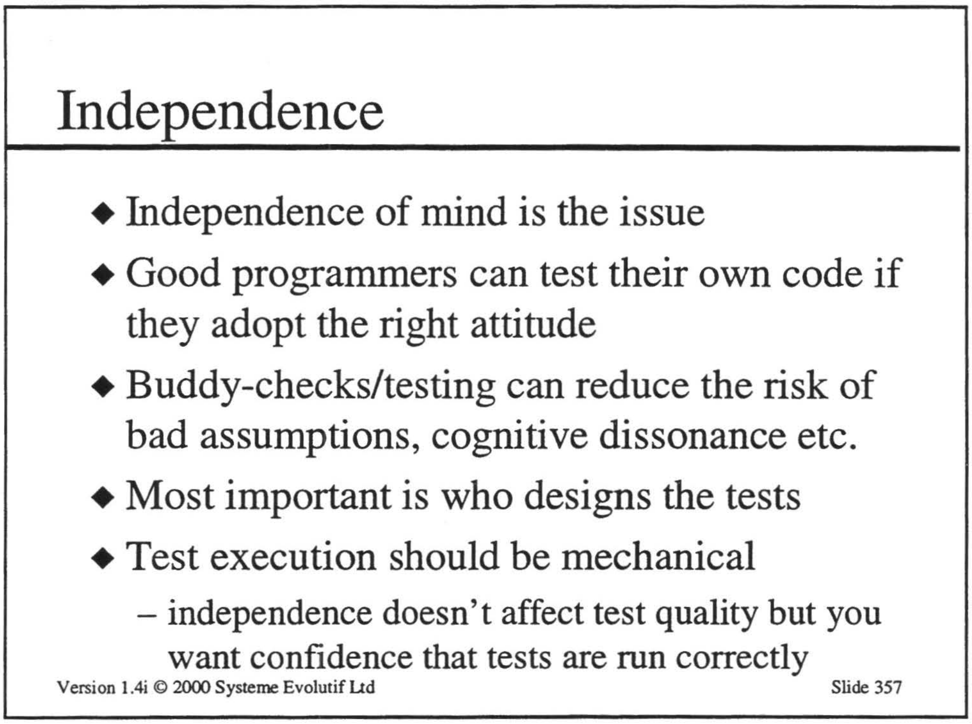


Who does the testing?

- ◆ Programmers do the ad-hoc testing
- ◆ Programmers, or other team member may do sub-system testing
- ◆ Independent teams usually do system testing
- ◆ Users (with support) do the users acceptance testing
- ◆ Independent organisations may be called upon to do any of the above testing formally.

Version 1.4i © 2000 Systeme Evolutif Ltd

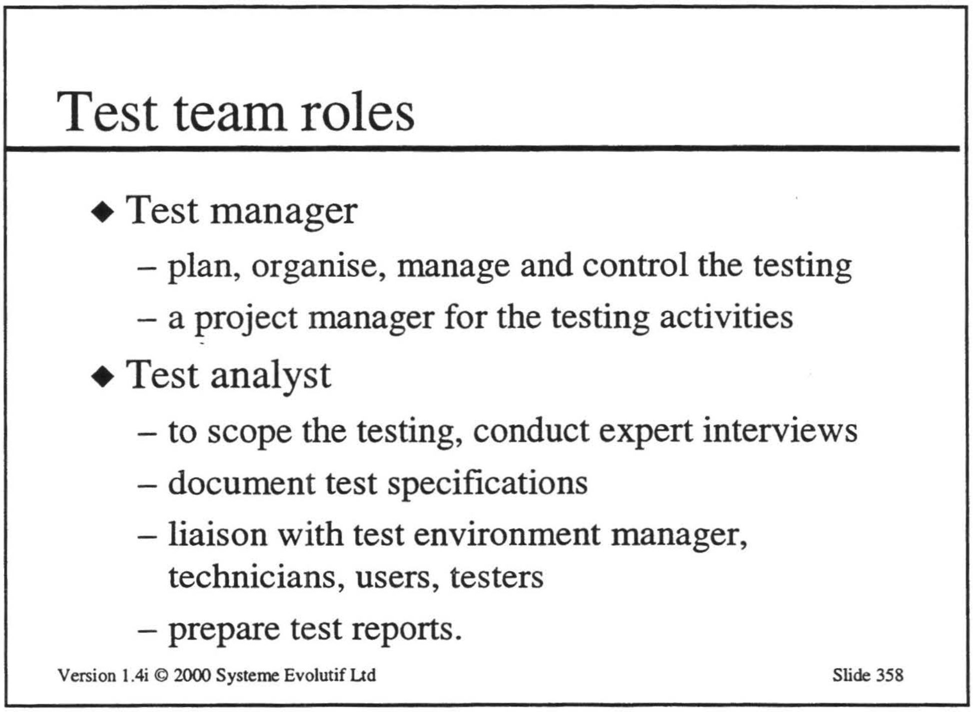
Slide 356



Independence

- ◆ Independence of mind is the issue
- ◆ Good programmers can test their own code if they adopt the right attitude
- ◆ Buddy-checks/testing can reduce the risk of bad assumptions, cognitive dissonance etc.
- ◆ Most important is who designs the tests
- ◆ Test execution should be mechanical
  - – independence doesn't affect test quality but you want confidence that tests are run correctly

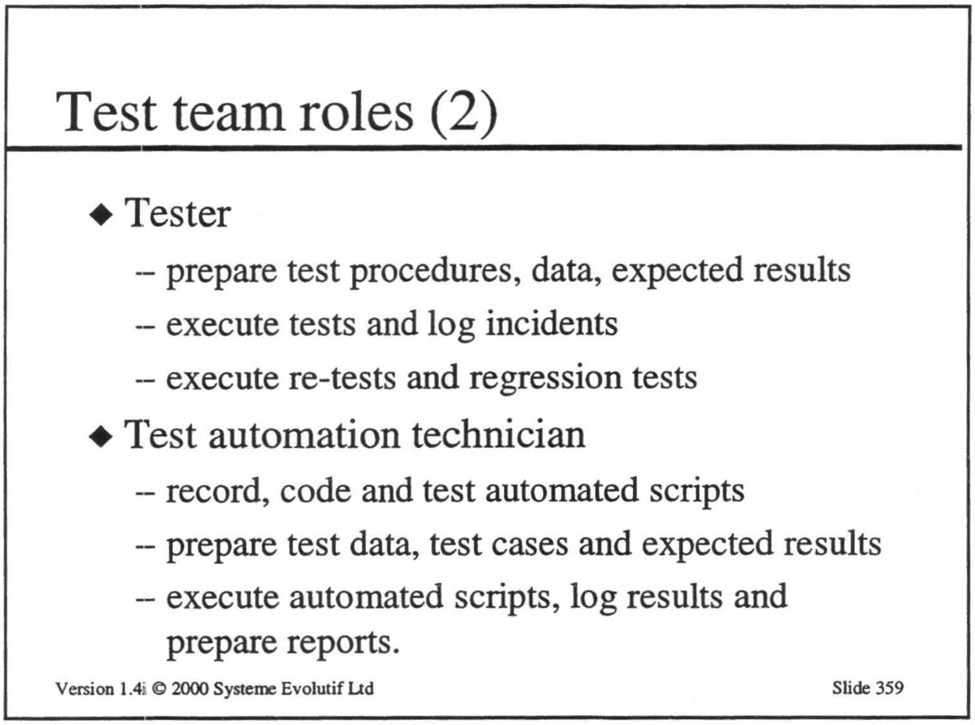
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 357



Test team roles

- ◆ Test manager
  - – plan, organise, manage and control the testing
  - – a project manager for the testing activities
- ◆ Test analyst
  - – to scope the testing, conduct expert interviews
  - – document test specifications
  - – liaison with test environment manager, technicians, users, testers
  - – prepare test reports.

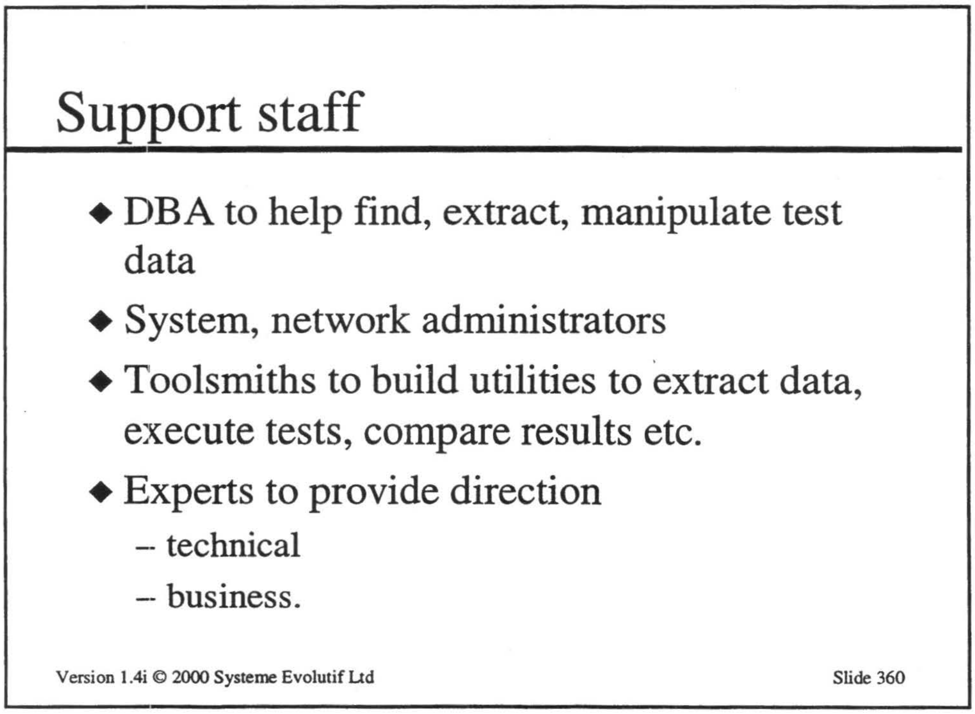
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 358



Test team roles (2)

- Tester
  - prepare test procedures, data, expected results
  - execute tests and log incidents
  - execute re-tests and regression tests
- Test automation technician
  - record, code and test automated scripts
  - prepare test data, test cases and expected results
  - execute automated scripts, log results and prepare reports.

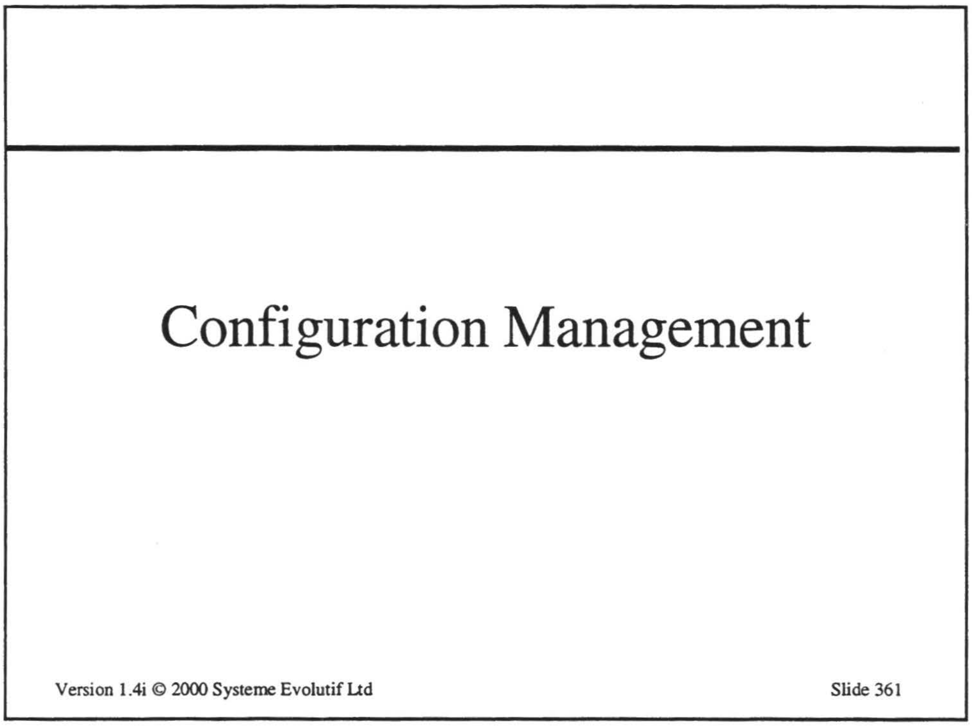
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 359



Support staff

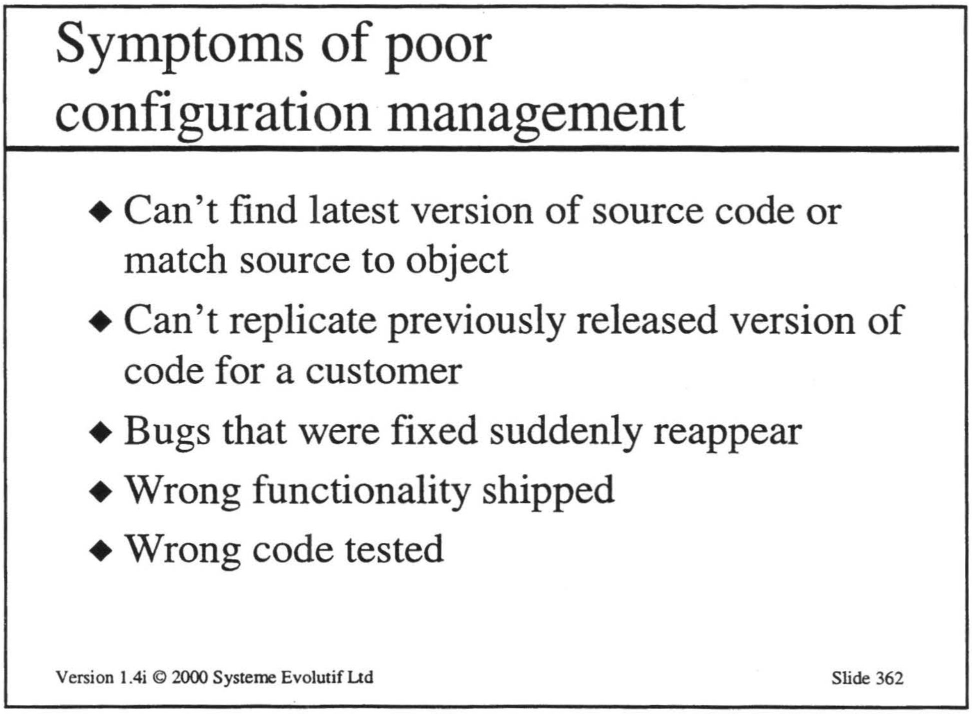
- DBA to help find, extract, manipulate test data
- System, network administrators
- Toolsmiths to build utilities to extract data, execute tests, compare results etc.
- Experts to provide direction
  - technical
  - business.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 360



Configuration Management

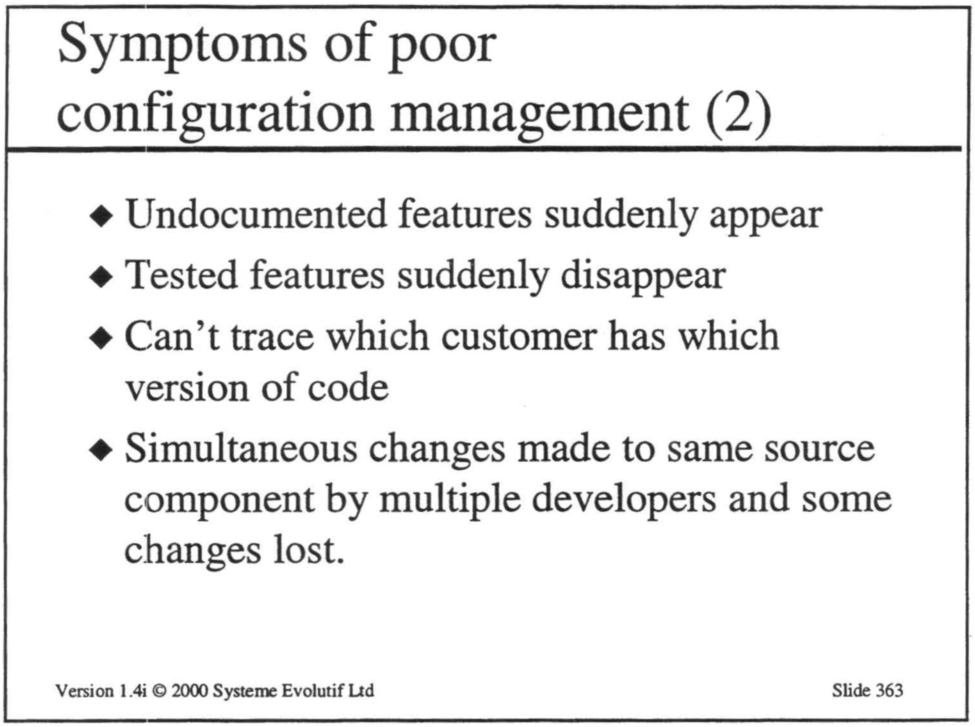
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 361



Symptoms of poor configuration management

- ◆ Can’t find latest version of source code or match source to object
- ◆ Can’t replicate previously released version of code for a customer
- ◆ Bugs that were fixed suddenly reappear
- ◆ Wrong functionality shipped
- ◆ Wrong code tested

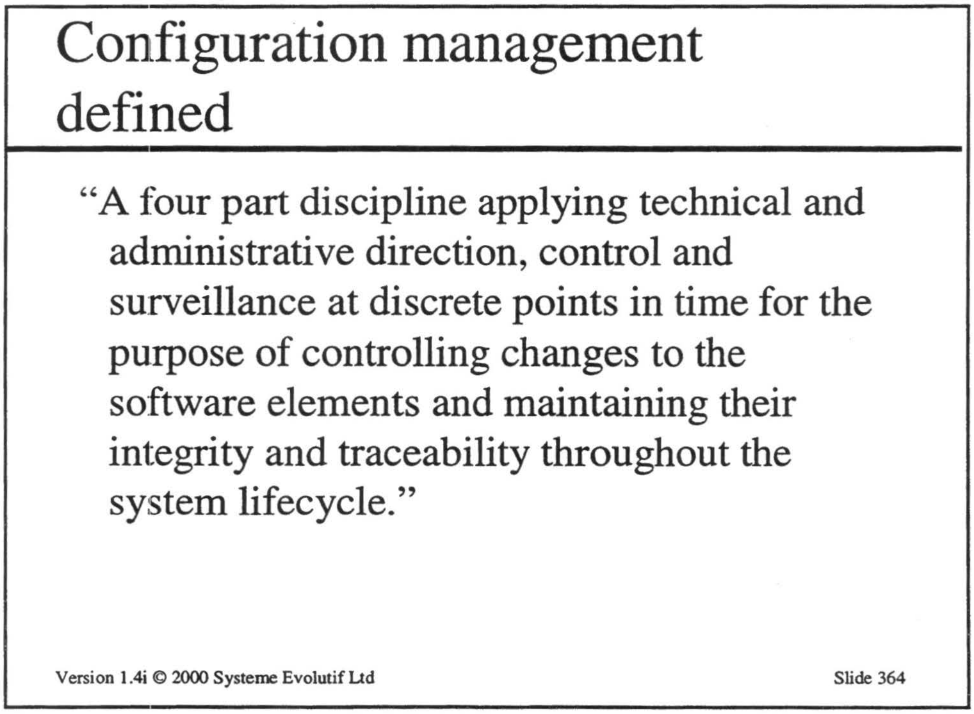
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 362



Symptoms of poor configuration management (2)

- ◆ Undocumented features suddenly appear
- ◆ Tested features suddenly disappear
- ◆ Can’t trace which customer has which version of code
- ◆ Simultaneous changes made to same source component by multiple developers and some changes lost.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 363



Configuration management defined

“A four part discipline applying technical and administrative direction, control and surveillance at discrete points in time for the purpose of controlling changes to the software elements and maintaining their integrity and traceability throughout the system lifecycle.”

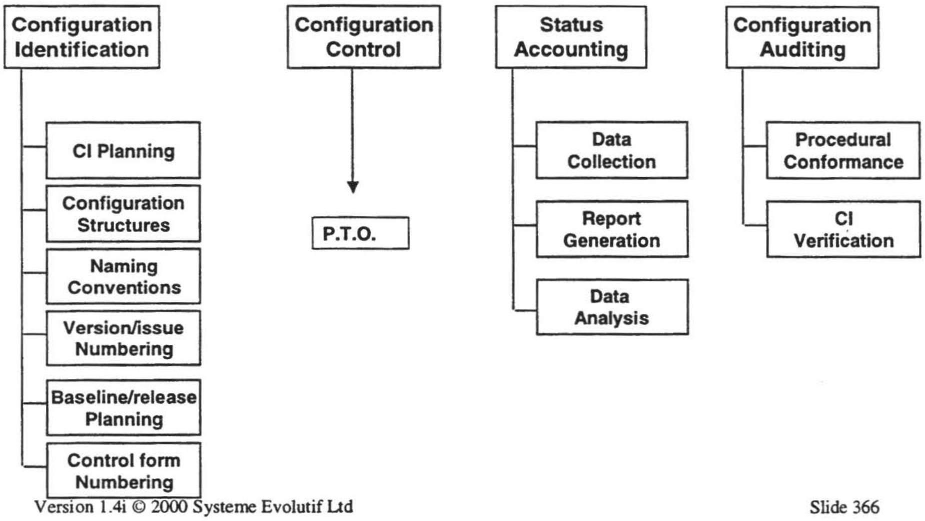
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 364

The answers CM provides

- What is our current software configuration?
- What is its status?
- How do we control changes to our configuration?
- What changes have been made to our software?
- Does anyone else's changes affect our software?

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 365

Software configuration management 1/2



Configuration Identification

- CI Planning
- Configuration Structures
- Naming Conventions
- Version/issue Numbering
- Baseline/release Planning
- Control form Numbering

Configuration Control

- P.T.O.

Status Accounting

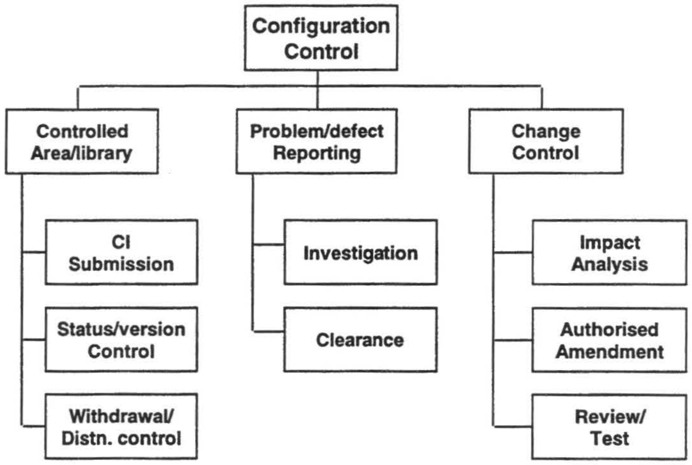
- Data Collection
- Report Generation
- Data Analysis

Configuration Auditing

- Procedural Conformance
- CI Verification

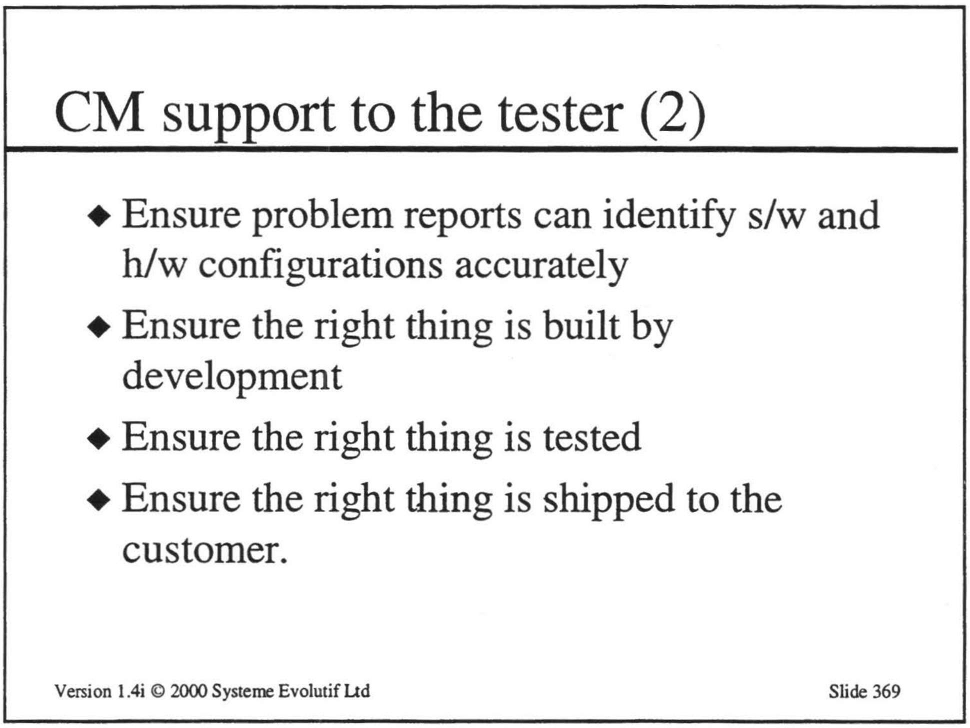
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 366

Software configuration management (2)



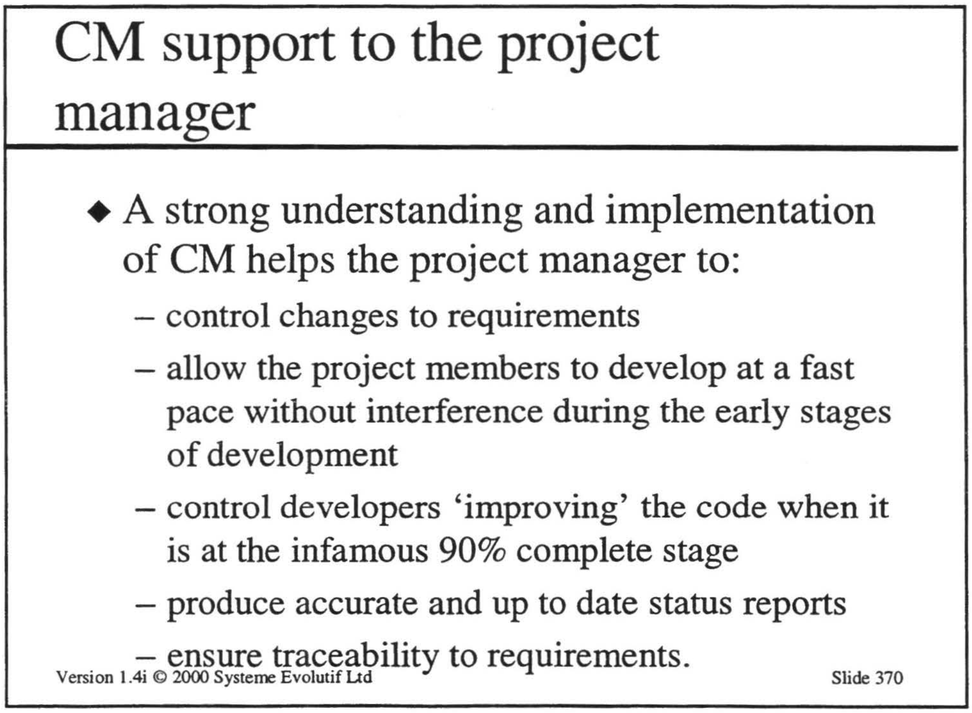
CM support to the tester

- A strong understanding and implementation of CM helps testers…
- Manage their own testware and their revision levels efficiently
- Associate a given version of a test with the appropriate version of the software to be tested
- Ensure traceability to requirements and problem reports.



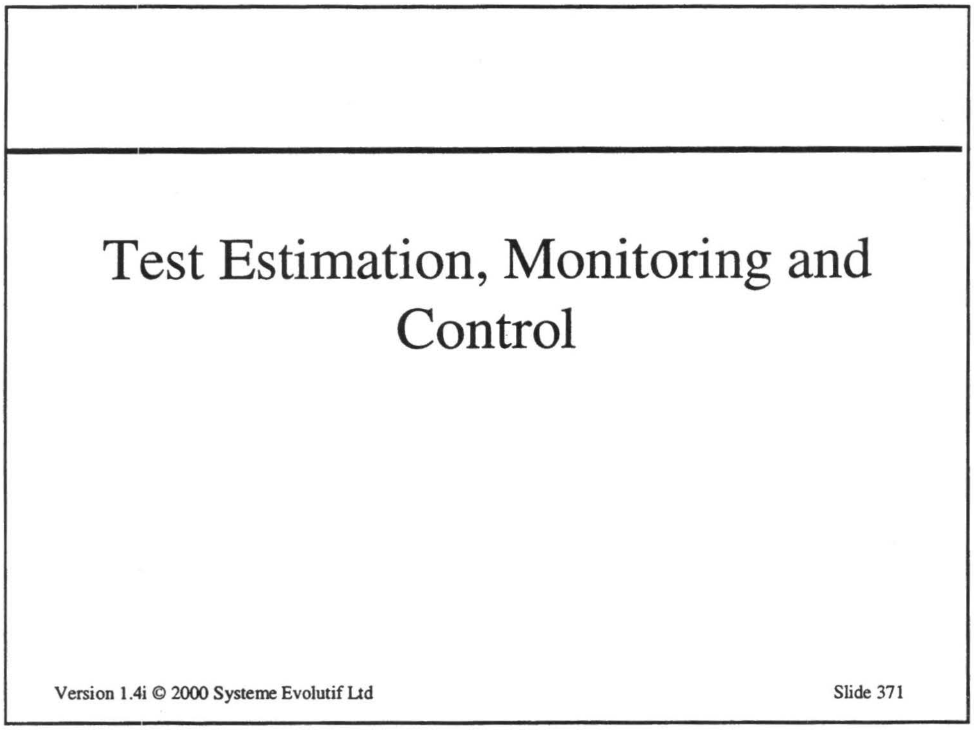
CM support to the tester (2)

- Ensure problem reports can identify s/w and h/w configurations accurately
- Ensure the right thing is built by development
- Ensure the right thing is tested
- Ensure the right thing is shipped to the customer.



CM support to the project manager

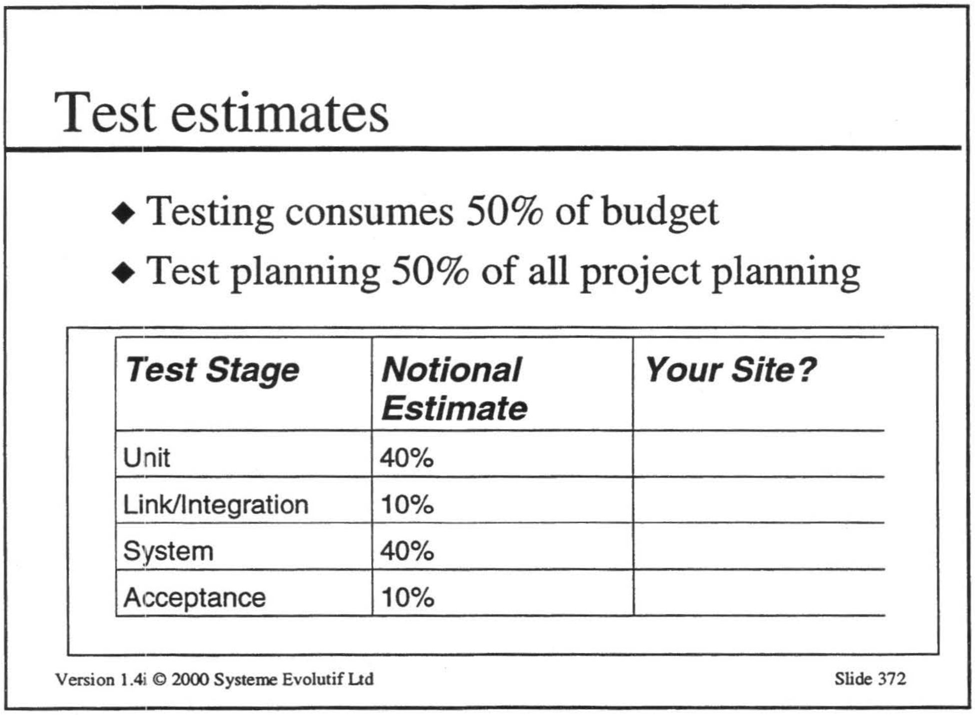
- A strong understanding and implementation of CM helps the project manager to:
  - control changes to requirements
  - allow the project members to develop at a fast pace without interference during the early stages of development
  - control developers 'improving' the code when it is at the infamous 90% complete stage
  - produce accurate and up to date status reports
  - ensure traceability to requirements.



Test Estimation, Monitoring and Control

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 371



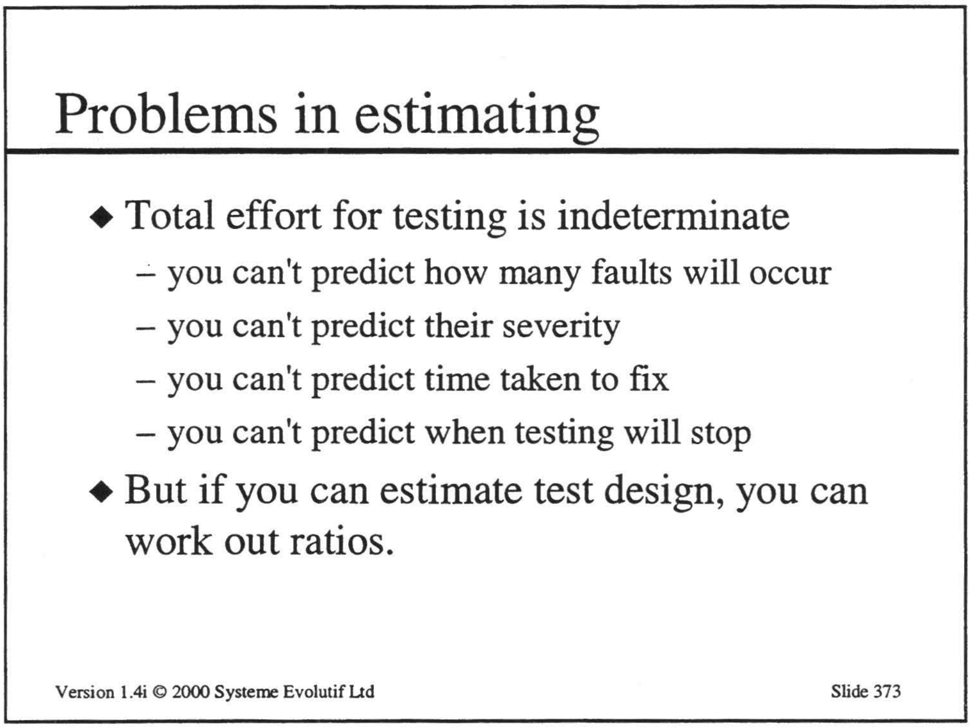
Test estimates

- ◆ Testing consumes 50% of budget
- ◆ Test planning 50% of all project planning

Test Stage	***Notional Estimate***	***Your Site?***
Unit	40%	
Link/Integration	10%	
System	40%	
Acceptance	10%	

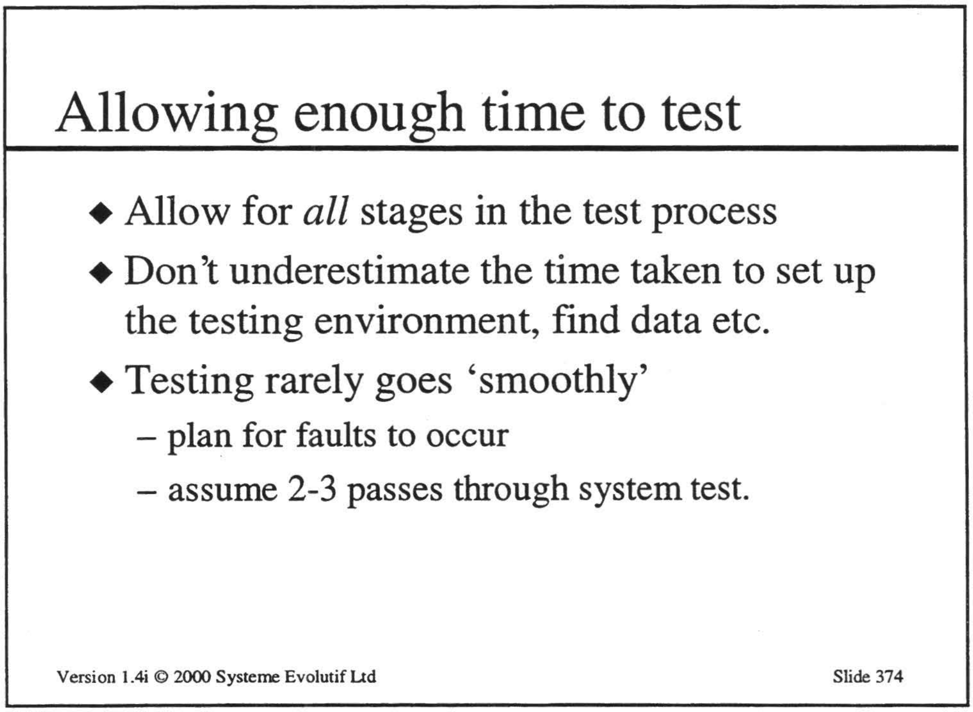
Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 372



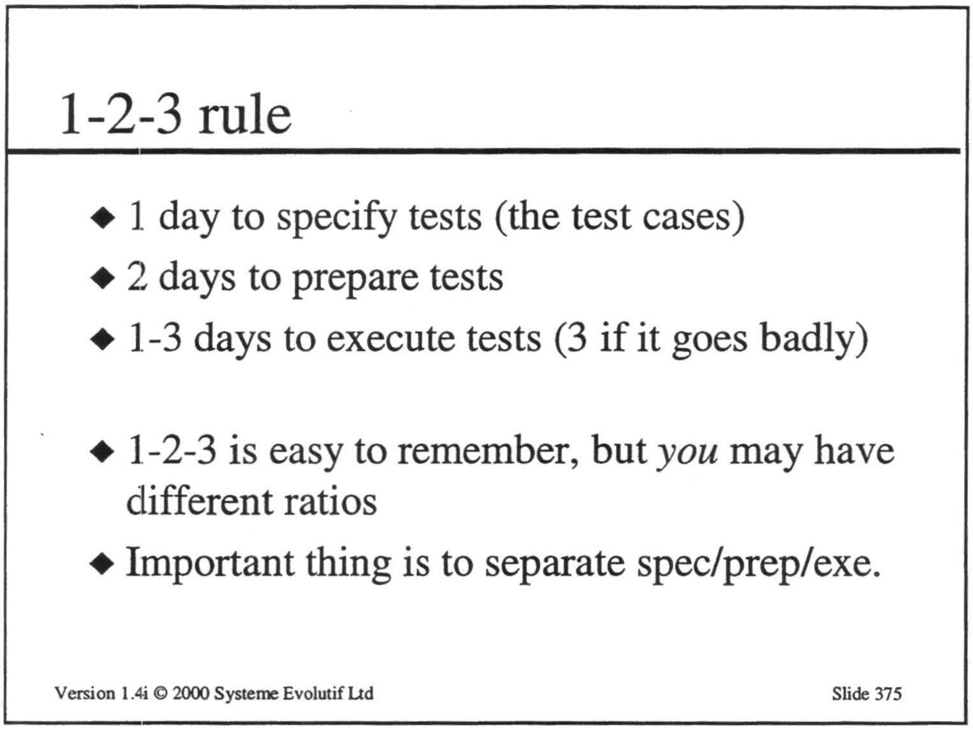
Problems in estimating

- Total effort for testing is indeterminate
  - you can't predict how many faults will occur
  - you can't predict their severity
  - you can't predict time taken to fix
  - you can't predict when testing will stop
- But if you can estimate test design, you can work out ratios.



Allowing enough time to test

- Allow for *all* stages in the test process
- Don't underestimate the time taken to set up the testing environment, find data etc.
- Testing rarely goes 'smoothly'
  - plan for faults to occur
  - assume 2-3 passes through system test.

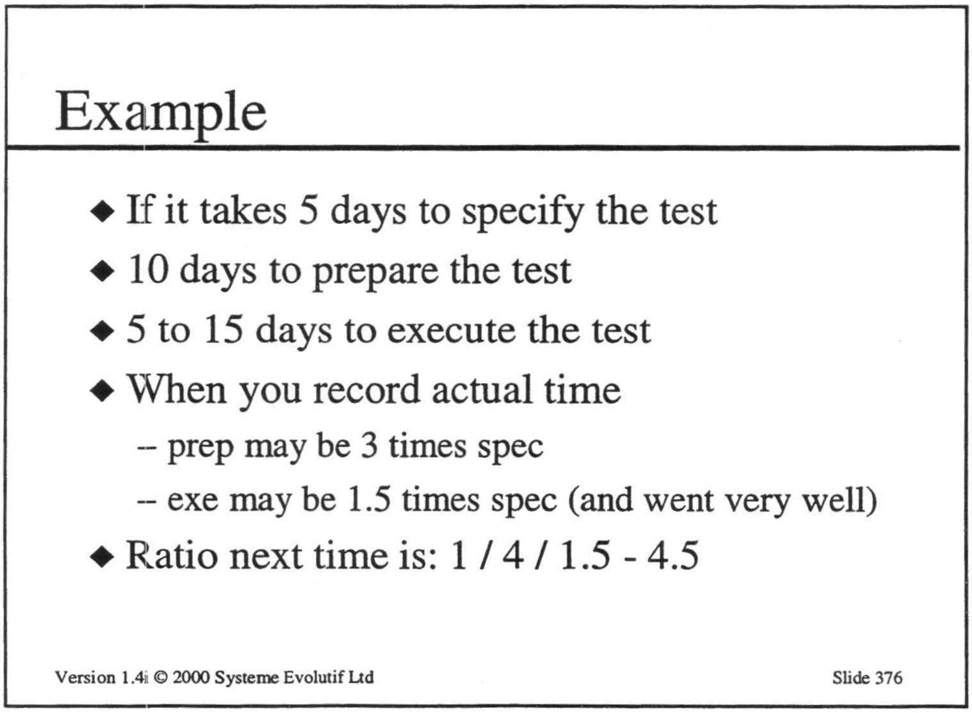


1-2-3 rule

- ◆ 1 day to specify tests (the test cases)
- ◆ 2 days to prepare tests
- ◆ 1-3 days to execute tests (3 if it goes badly)

- ◆ 1-2-3 is easy to remember, but *you* may have different ratios
- ◆ Important thing is to separate spec/prep/exe.

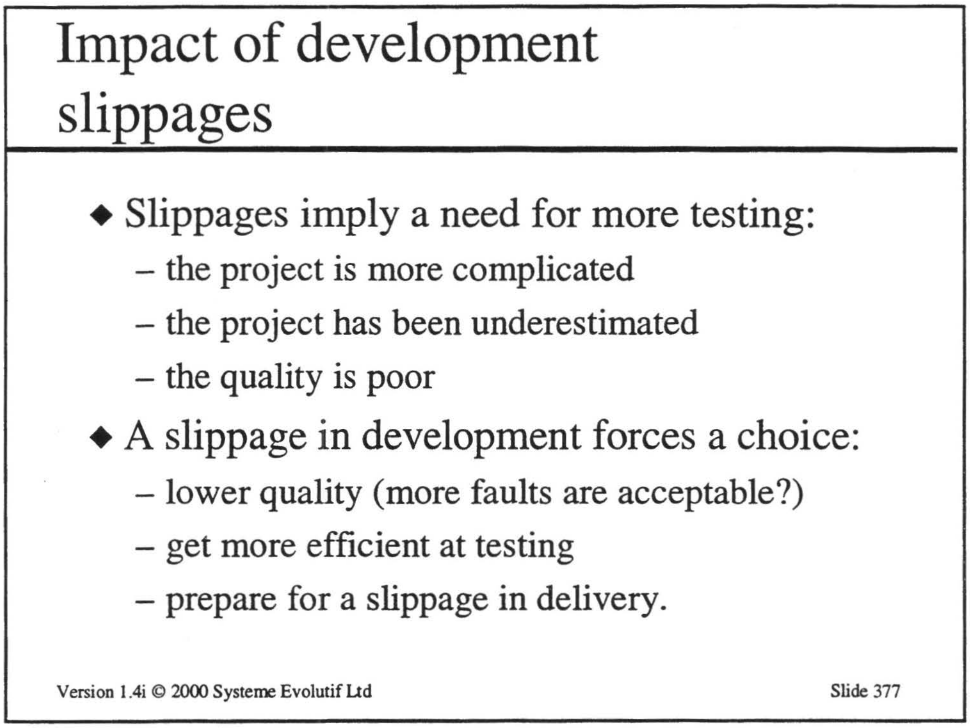
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 375



Example

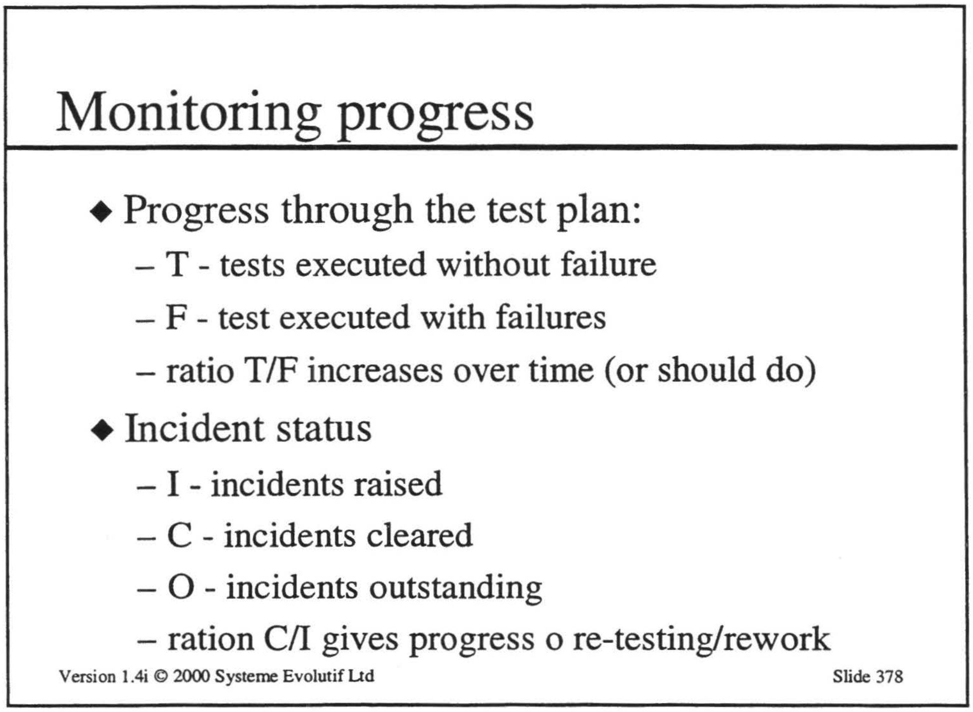
- ◆ If it takes 5 days to specify the test
- ◆ 10 days to prepare the test
- ◆ 5 to 15 days to execute the test
- ◆ When you record actual time
  - – prep may be 3 times spec
  - – exe may be 1.5 times spec (and went very well)
- ◆ Ratio next time is: 1 / 4 / 1.5 - 4.5

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 376



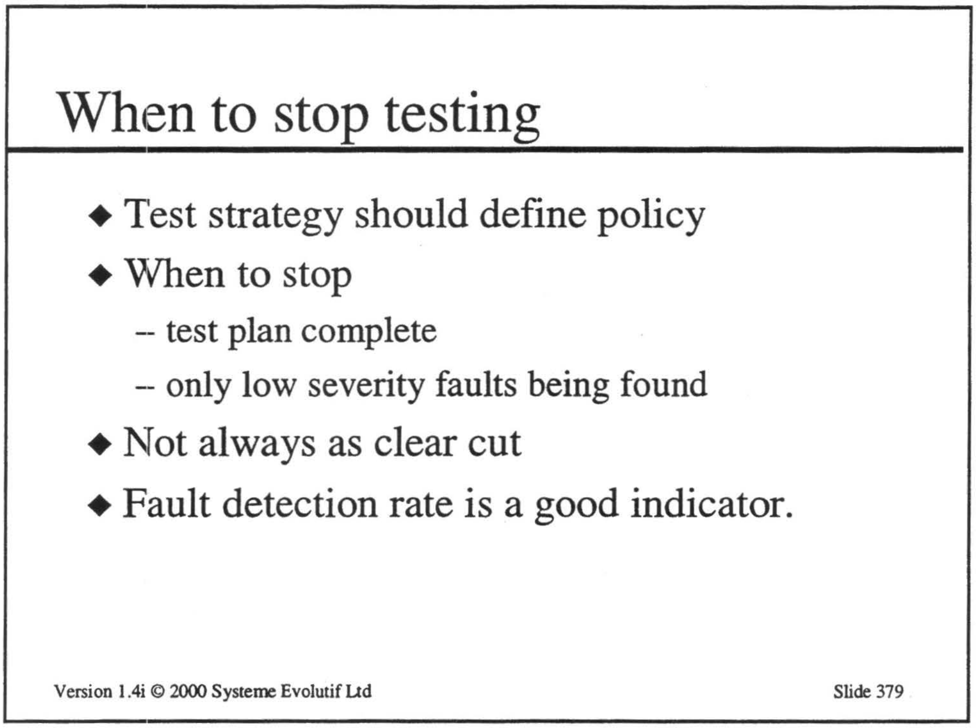
Impact of development slippages

- Slippages imply a need for more testing:
  - the project is more complicated
  - the project has been underestimated
  - the quality is poor
- A slippage in development forces a choice:
  - lower quality (more faults are acceptable?)
  - get more efficient at testing
  - prepare for a slippage in delivery.



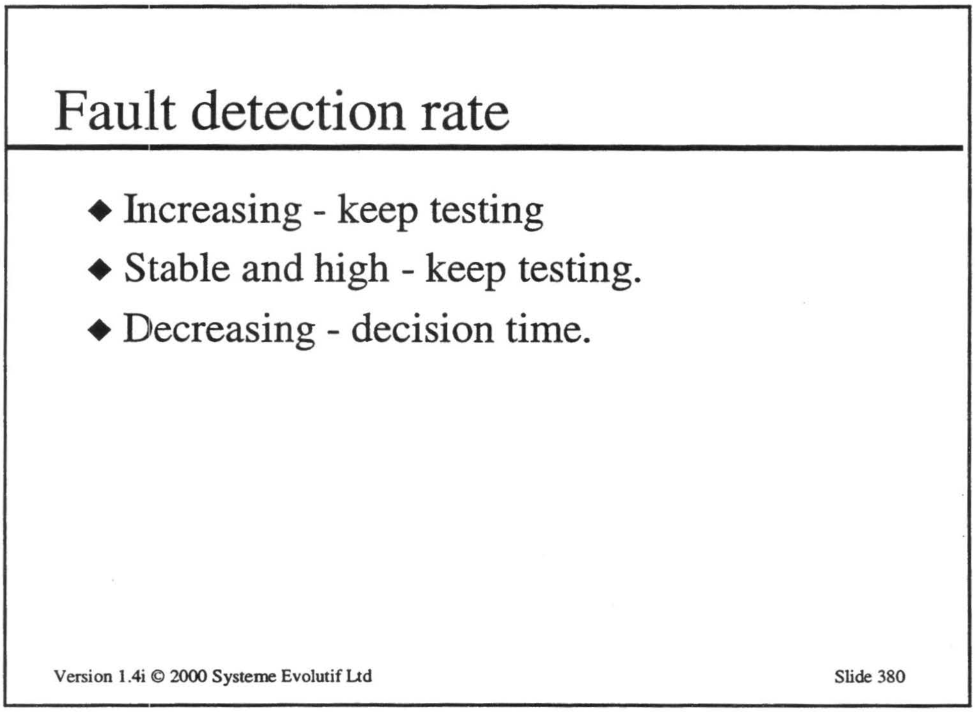
Monitoring progress

- Progress through the test plan:
  - T - tests executed without failure
  - F - test executed with failures
  - ratio T/F increases over time (or should do)
- Incident status
  - I - incidents raised
  - C - incidents cleared
  - O - incidents outstanding
  - ration C/I gives progress o re-testing/rework



When to stop testing

- ◆ Test strategy should define policy
- ◆ When to stop
  - -- test plan complete
  - -- only low severity faults being found
- ◆ Not always as clear cut
- ◆ Fault detection rate is a good indicator.

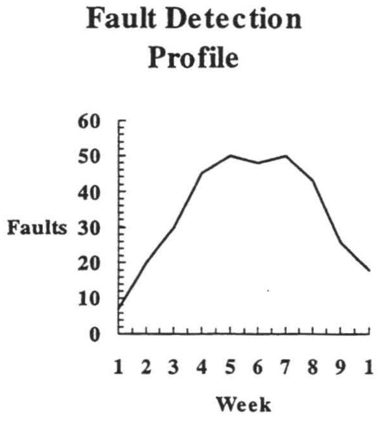


Fault detection rate

- ◆ Increasing - keep testing
- ◆ Stable and high - keep testing.
- ◆ Decreasing - decision time.

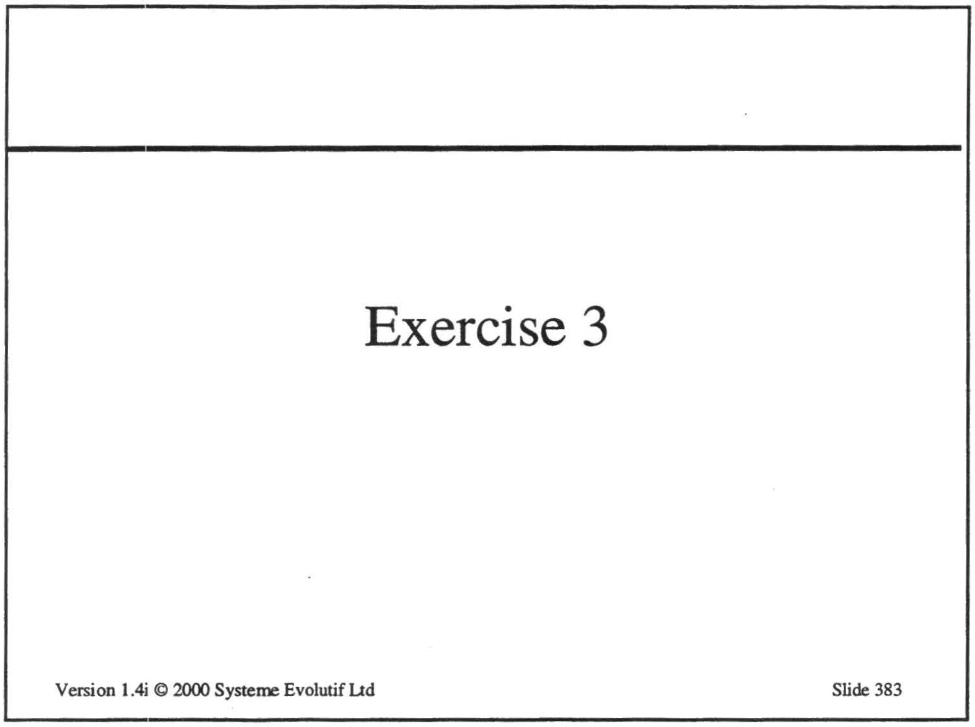
Fault detection rate (2)

- Detection rate is high and stable until week 8 - don't consider stopping until then
- Look at severity of faults being found
- Don't lose sight of your overall target though.



Running out of time

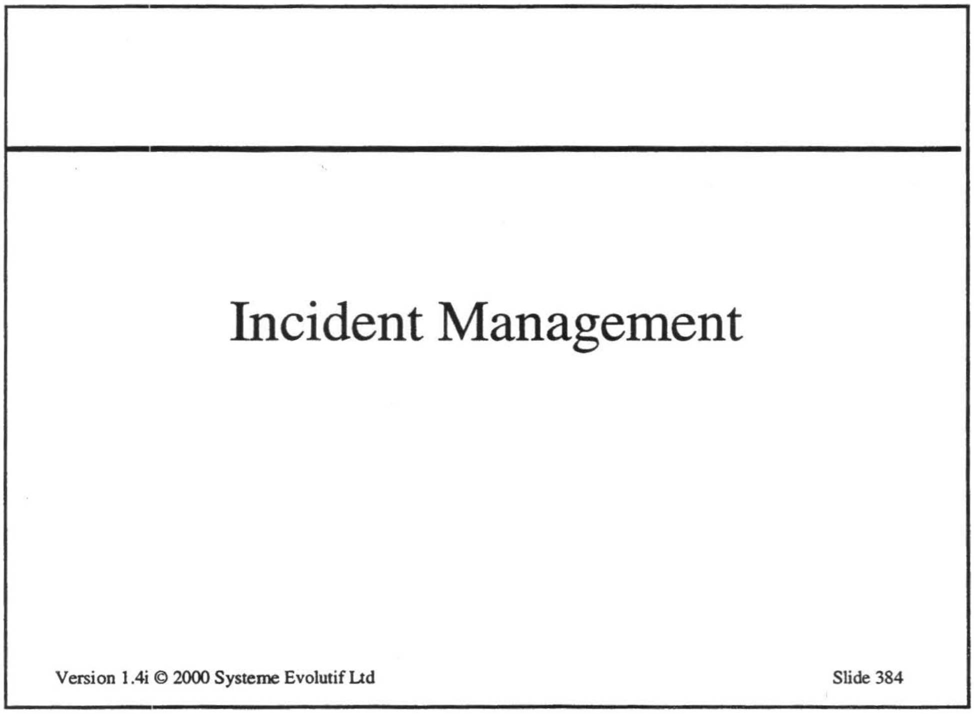
- The time has come to stop testing
- If you have additional test to run, what are the risks of not running before release?
- What are the severity of outstanding faults?
  - would they preclude release?
  - would they be acceptable?
- Can you continue testing, but release anyway?



Exercise 3

Version 1.4i © 2000 Systeme Evolutif Ltd

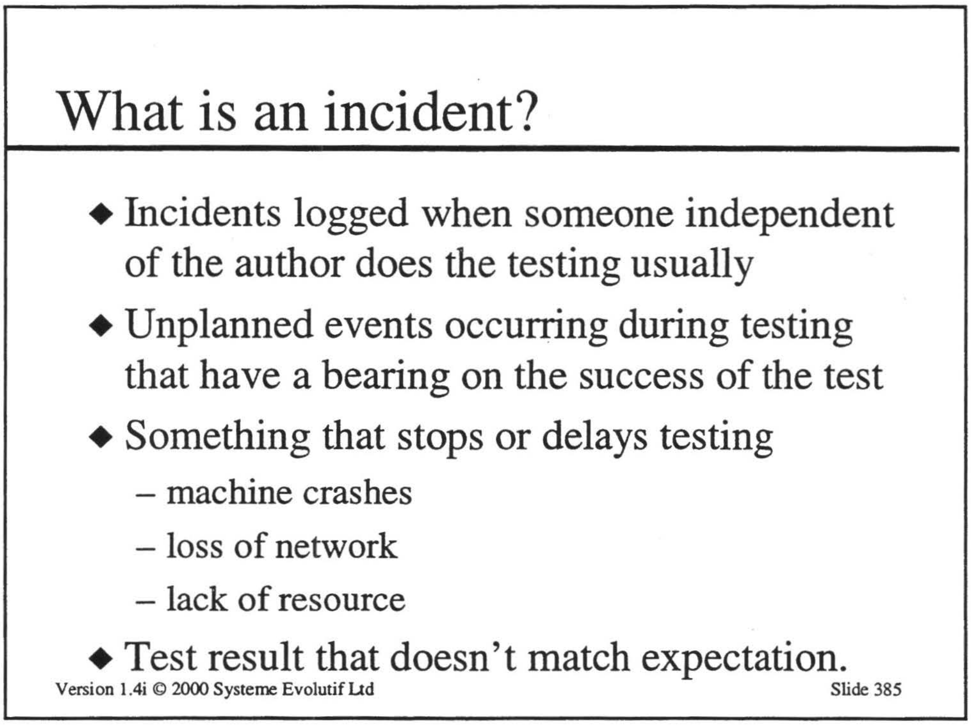
Slide 383



Incident Management

Version 1.4i © 2000 Systeme Evolutif Ltd

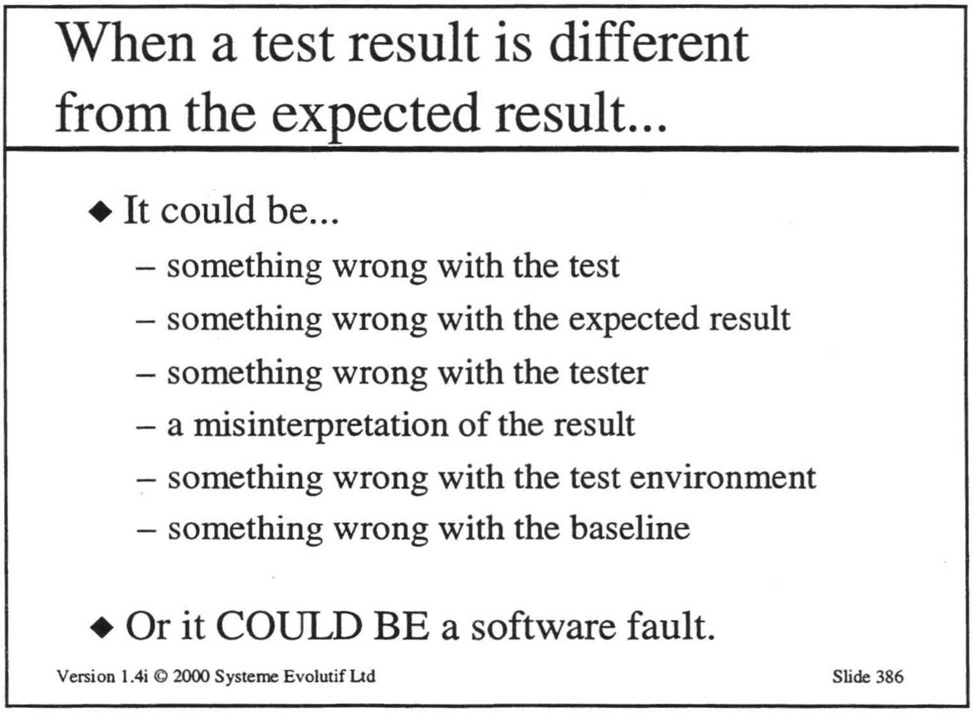
Slide 384



What is an incident?

- ◆ Incidents logged when someone independent of the author does the testing usually
- ◆ Unplanned events occurring during testing that have a bearing on the success of the test
- ◆ Something that stops or delays testing
  - – machine crashes
  - – loss of network
  - – lack of resource
- ◆ Test result that doesn't match expectation.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 385



When a test result is different from the expected result...

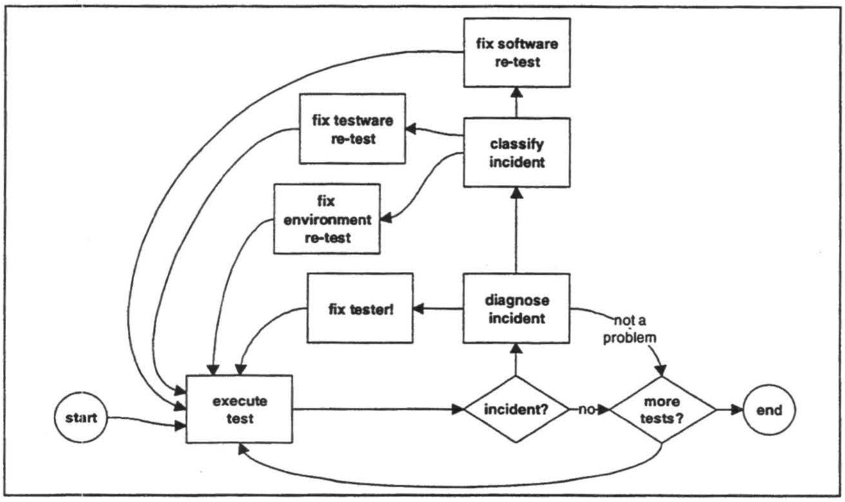
- ◆ It could be...
  - – something wrong with the test
  - – something wrong with the expected result
  - – something wrong with the tester
  - – a misinterpretation of the result
  - – something wrong with the test environment
  - – something wrong with the baseline
- ◆ Or it COULD BE a software fault.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 386

Incident logging

- Tester should stop and complete an incident log
  - describe exactly what is "wrong"
  - record the test script and script step
  - attach any outputs (screen dumps or printouts) that may be useful
  - impact on your current tests
    - » stops all testing, stops current test, no impact...

Typical test execution and incident management process



fix software re-test

fix testware re-test

classify incident

fix environment re-test

fix tester!

diagnose incident

not a problem

start

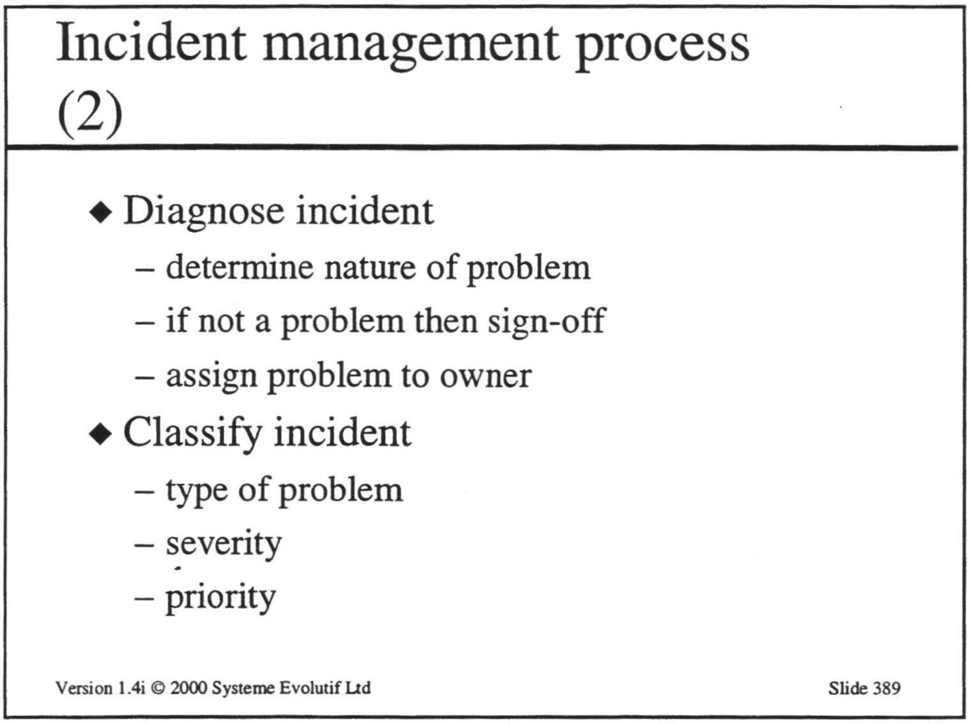
execute test

incident?

no

more tests?

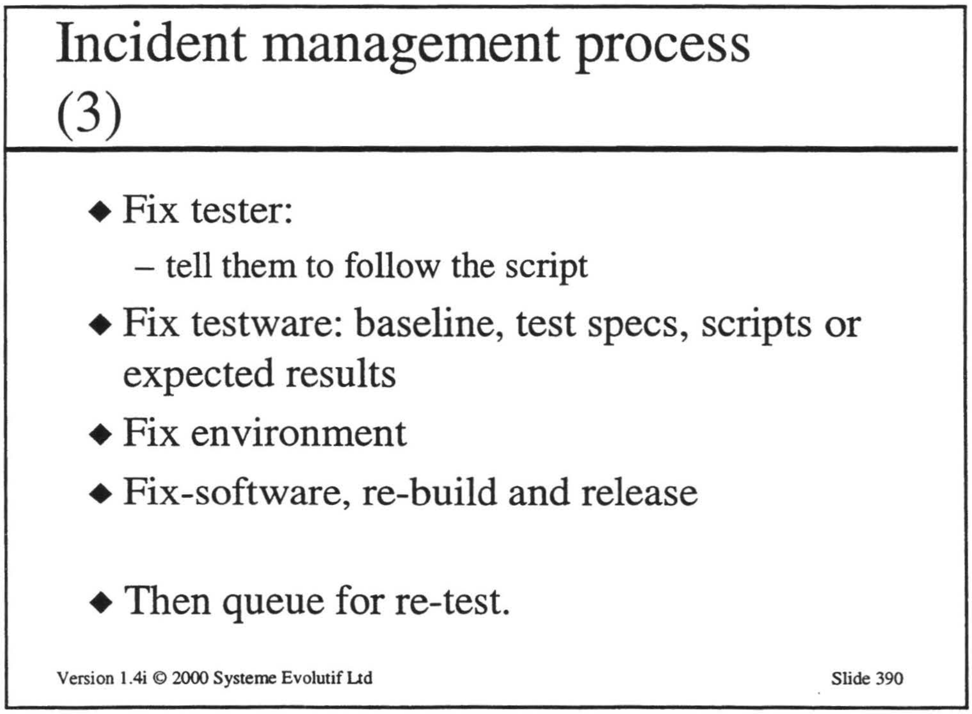
end



Incident management process (2)

- ◆ Diagnose incident
  - – determine nature of problem
  - – if not a problem then sign-off
  - – assign problem to owner
- ◆ Classify incident
  - – type of problem
  - – severity
  - – priority

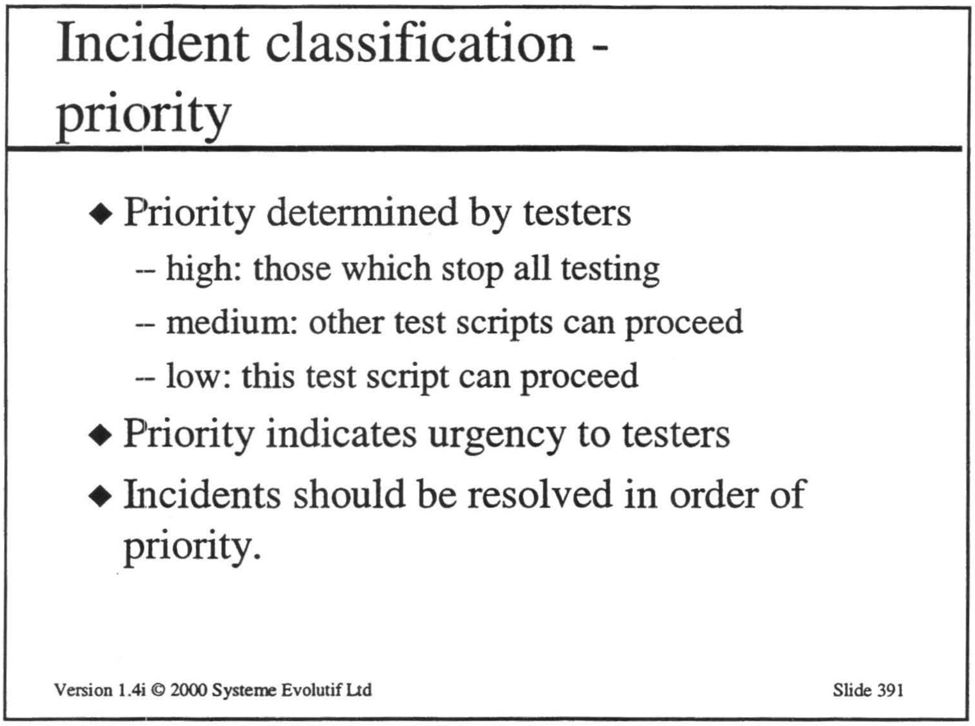
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 389



Incident management process (3)

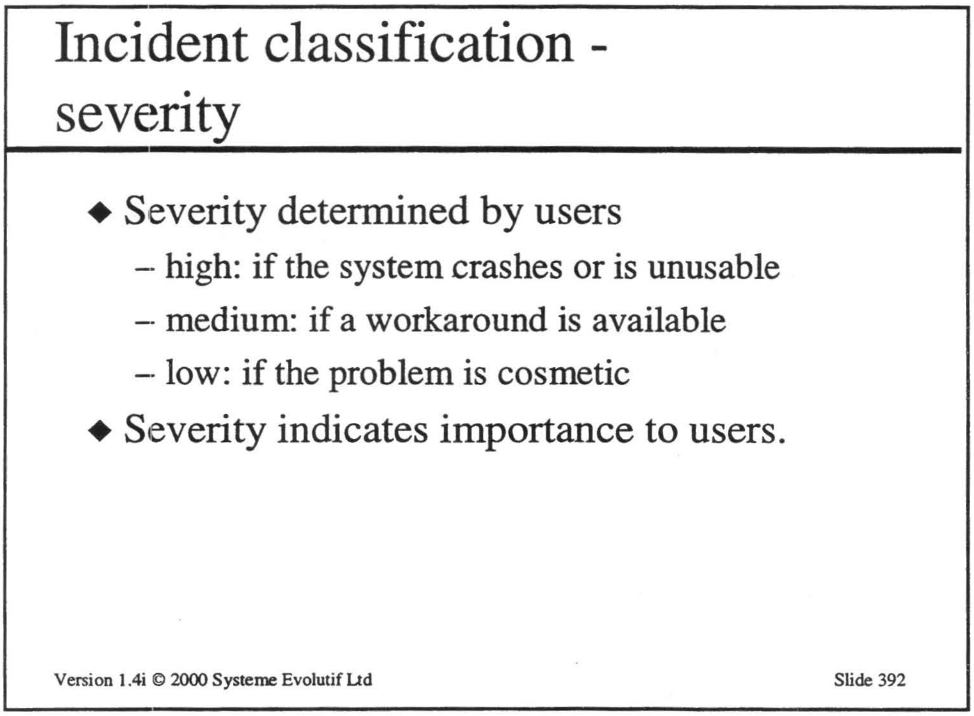
- ◆ Fix tester:
  - – tell them to follow the script
- ◆ Fix testware: baseline, test specs, scripts or expected results
- ◆ Fix environment
- ◆ Fix-software, re-build and release
- ◆ Then queue for re-test.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 390



Incident classification - priority

- Priority determined by testers
  - high: those which stop all testing
  - medium: other test scripts can proceed
  - low: this test script can proceed
- Priority indicates urgency to testers
- Incidents should be resolved in order of priority.



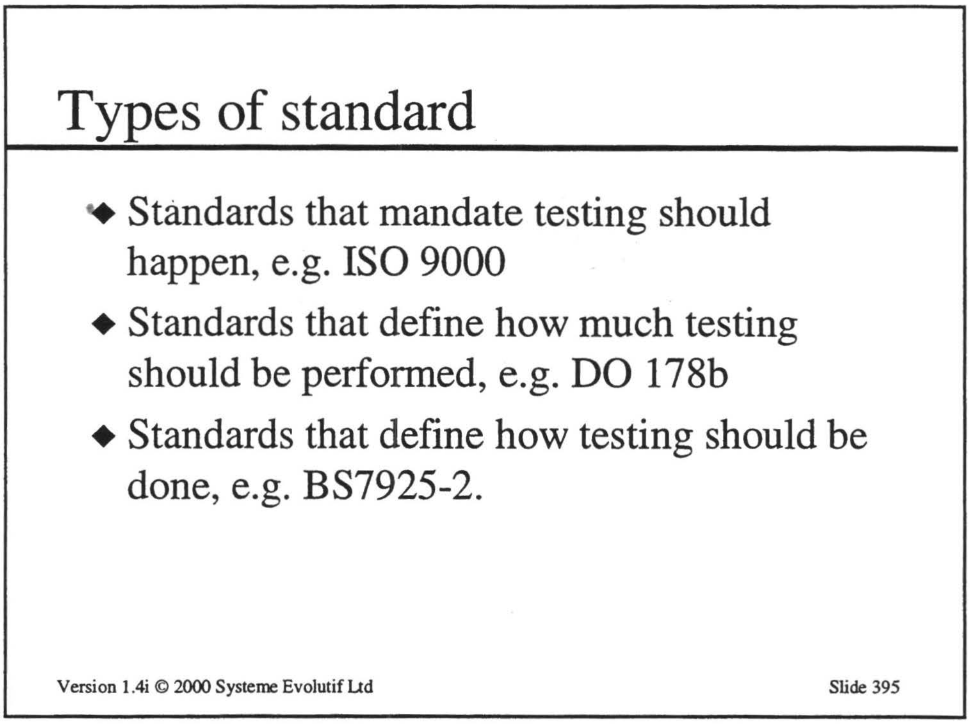
Incident classification - severity

- Severity determined by users
  - high: if the system crashes or is unusable
  - medium: if a workaround is available
  - low: if the problem is cosmetic
- Severity indicates importance to users.

Software fixing

- The developers must have enough information to reproduce the problem
- If developers can’t reproduce it, they probably can’t fix it
  - try and reproduce it yourself, if possible
  - make sure your description in the incident report is adequate
- Incidents get prioritised and developer resources get assigned according to priority.

Standards for Testing

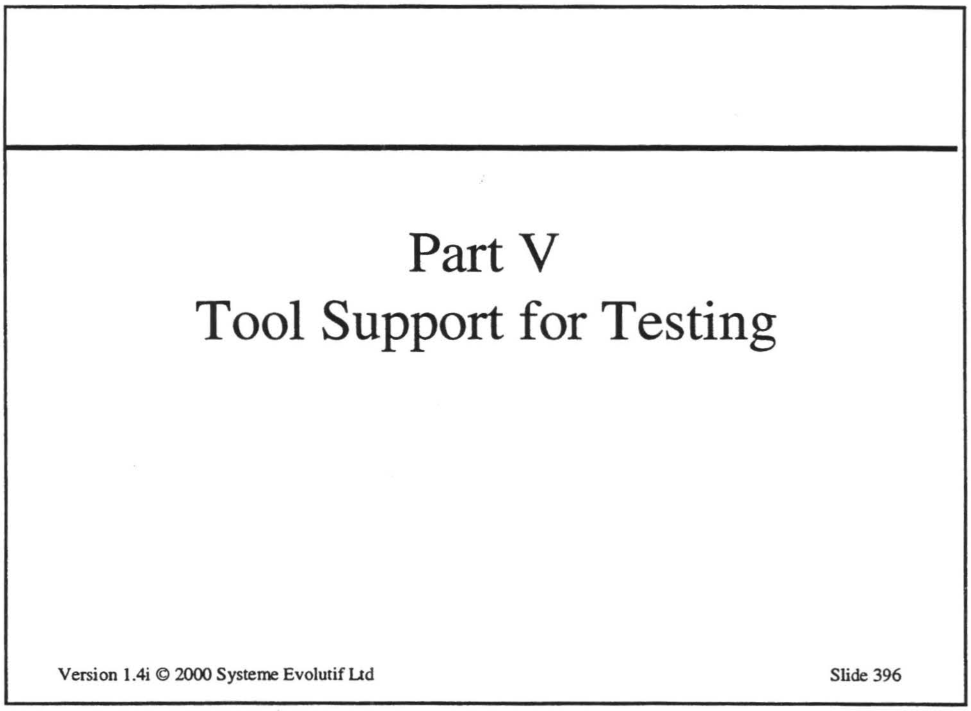


Types of standard

- Standards that mandate testing should happen, e.g. ISO 9000
- Standards that define how much testing should be performed, e.g. DO 178b
- Standards that define how testing should be done, e.g. BS7925-2.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 395



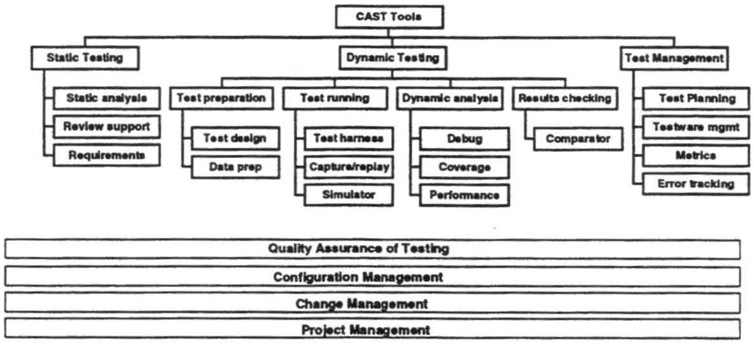
Part V
Tool Support for Testing

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 396

Types of CAST Tool

Categories of CAST tools



CAST Tools

Static Testing
- Static analysis
- Review support
- Requirements

Dynamic Testing
- Test preparation
  - Test design
  - Data prep
- Test running
  - Test harness
  - Capture/replay
  - Simulator
- Dynamic analysis
  - Debug
  - Coverage
  - Performance
- Results checking
  - Comparator

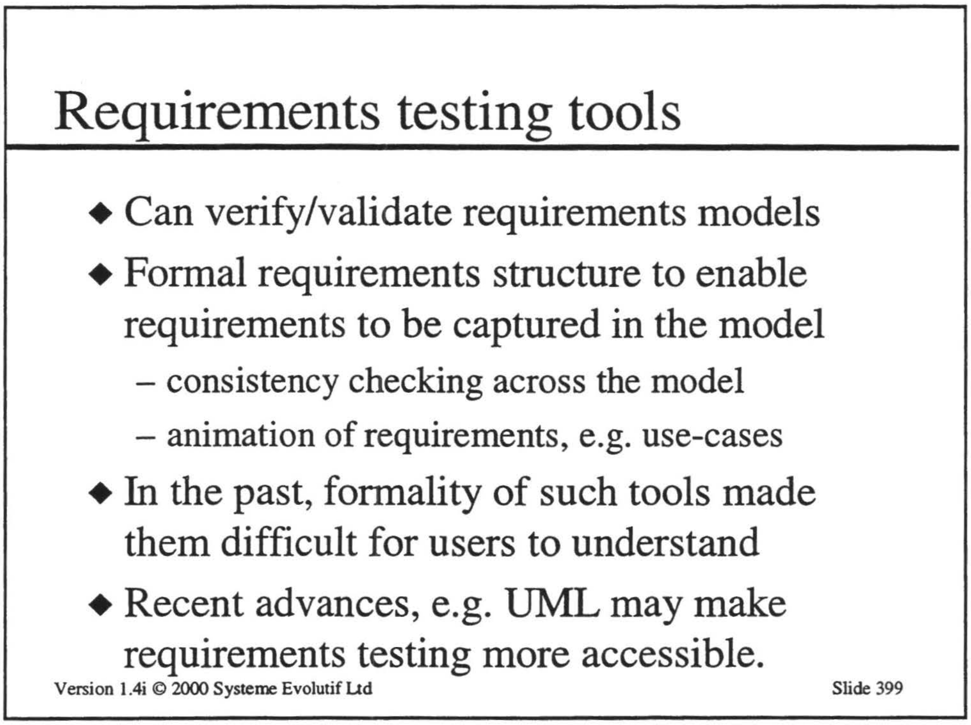
Test Management
- Test Planning
- Testware mgmt
- Metrics
- Error tracking

Quality Assurance of Testing

Configuration Management

Change Management

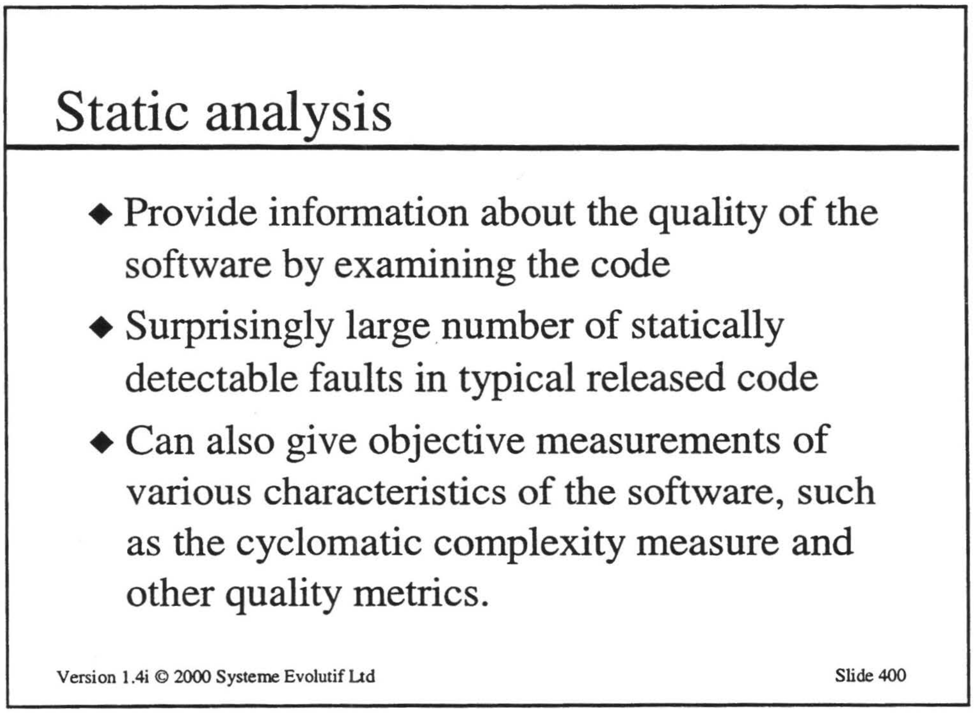
Project Management



Requirements testing tools

- Can verify/validate requirements models
- Formal requirements structure to enable requirements to be captured in the model
  - consistency checking across the model
  - animation of requirements, e.g. use-cases
- In the past, formality of such tools made them difficult for users to understand
- Recent advances, e.g. UML may make requirements testing more accessible.

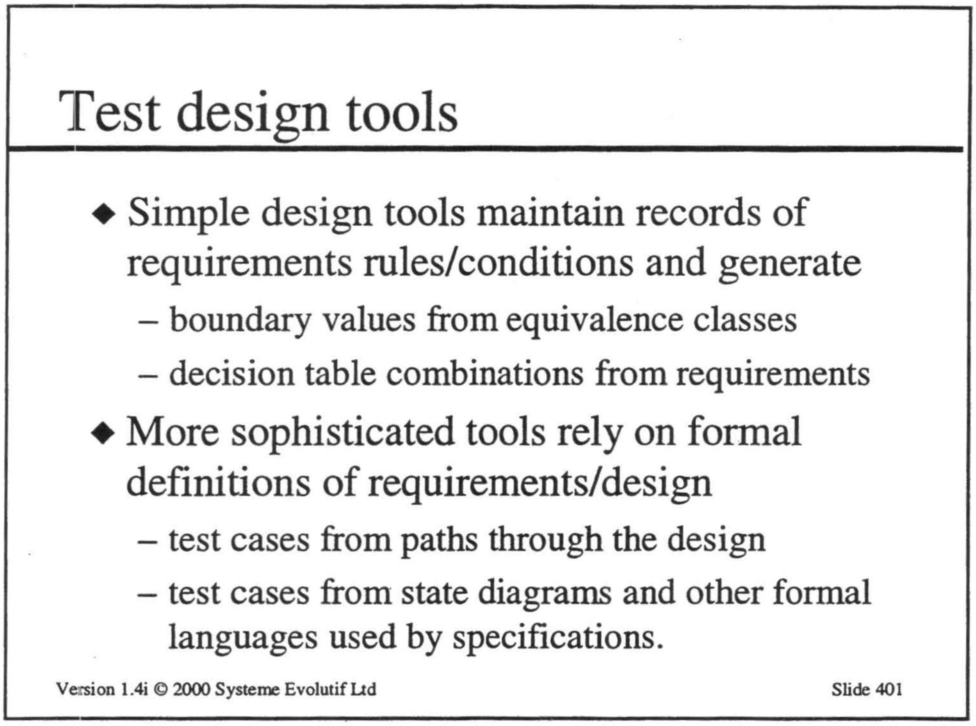
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 399



Static analysis

- Provide information about the quality of the software by examining the code
- Surprisingly large number of statically detectable faults in typical released code
- Can also give objective measurements of various characteristics of the software, such as the cyclomatic complexity measure and other quality metrics.

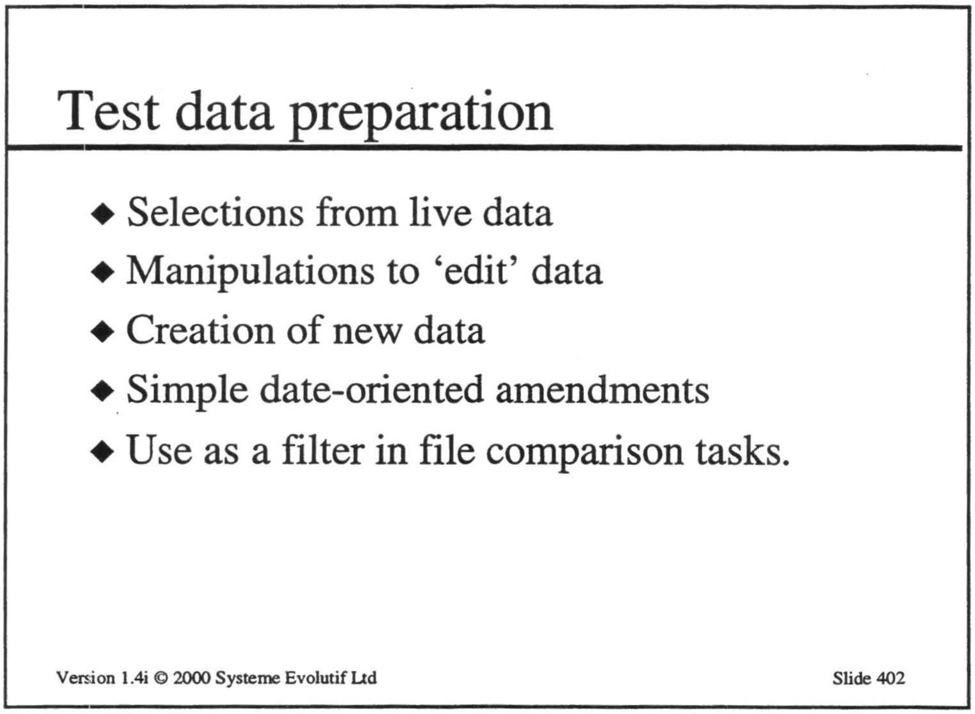
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 400



Test design tools

- ◆ Simple design tools maintain records of requirements rules/conditions and generate
  - – boundary values from equivalence classes
  - – decision table combinations from requirements
- ◆ More sophisticated tools rely on formal definitions of requirements/design
  - – test cases from paths through the design
  - – test cases from state diagrams and other formal languages used by specifications.

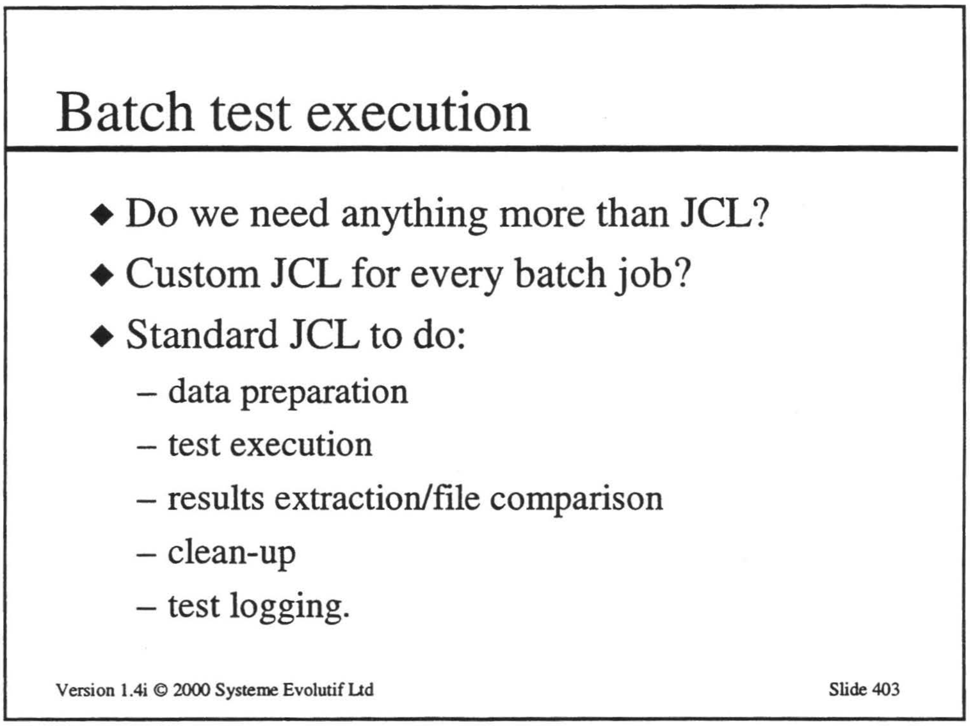
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 401



Test data preparation

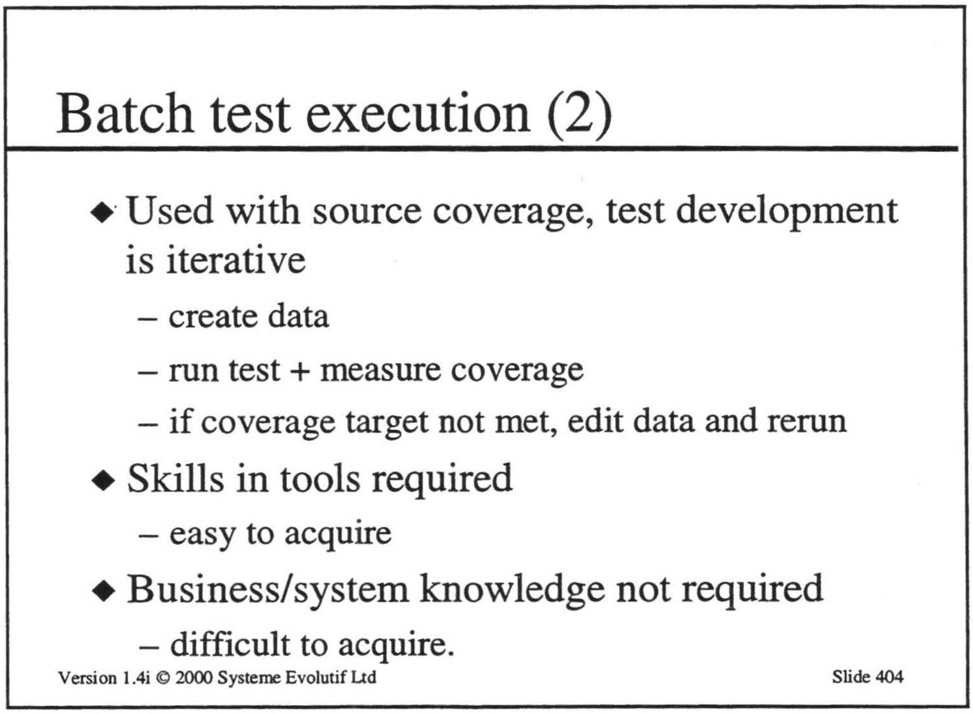
- ◆ Selections from live data
- ◆ Manipulations to 'edit' data
- ◆ Creation of new data
- ◆ Simple date-oriented amendments
- ◆ Use as a filter in file comparison tasks.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 402



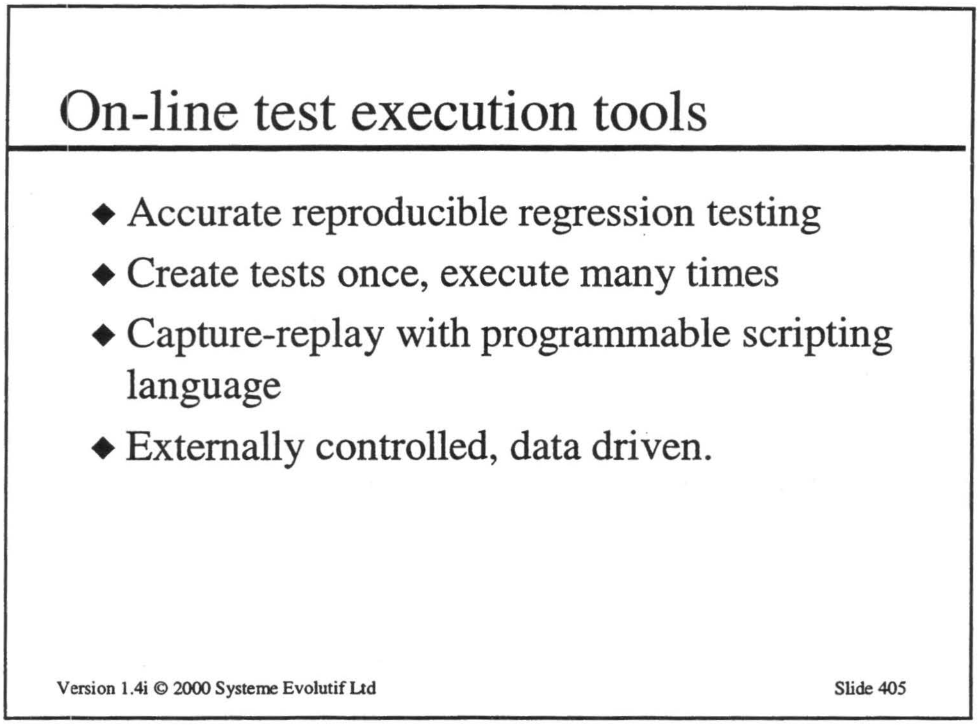
Batch test execution

- Do we need anything more than JCL?
- Custom JCL for every batch job?
- Standard JCL to do:
  - data preparation
  - test execution
  - results extraction/file comparison
  - clean-up
  - test logging.



Batch test execution (2)

- Used with source coverage, test development is iterative
  - create data
  - run test + measure coverage
  - if coverage target not met, edit data and rerun
- Skills in tools required
  - easy to acquire
- Business/system knowledge not required
  - difficult to acquire.

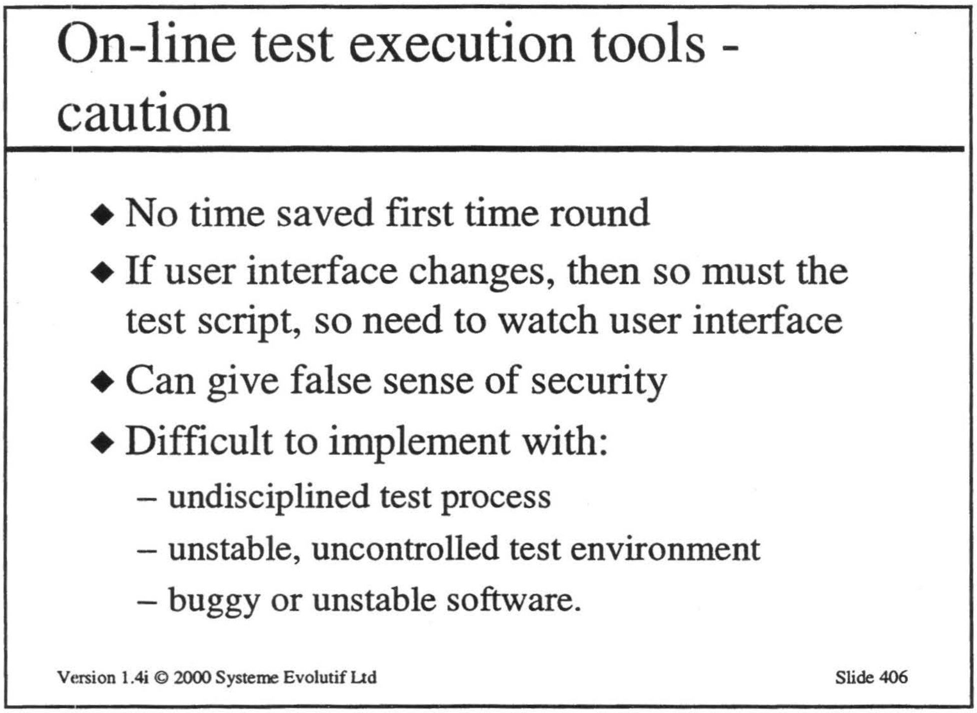


On-line test execution tools

- Accurate reproducible regression testing
- Create tests once, execute many times
- Capture-replay with programmable scripting language
- Externally controlled, data driven.

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 405

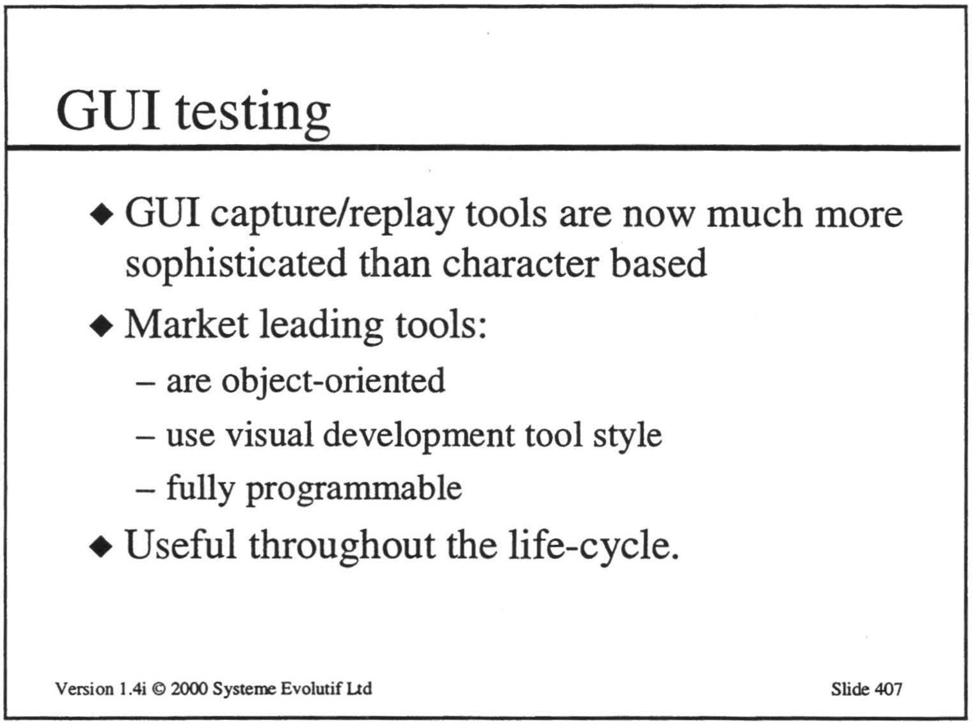


On-line test execution tools - caution

- No time saved first time round
- If user interface changes, then so must the test script, so need to watch user interface
- Can give false sense of security
- Difficult to implement with:
  - – undisciplined test process
  - – unstable, uncontrolled test environment
  - – buggy or unstable software.

Version 1.4i © 2000 Systeme Evolutif Ltd

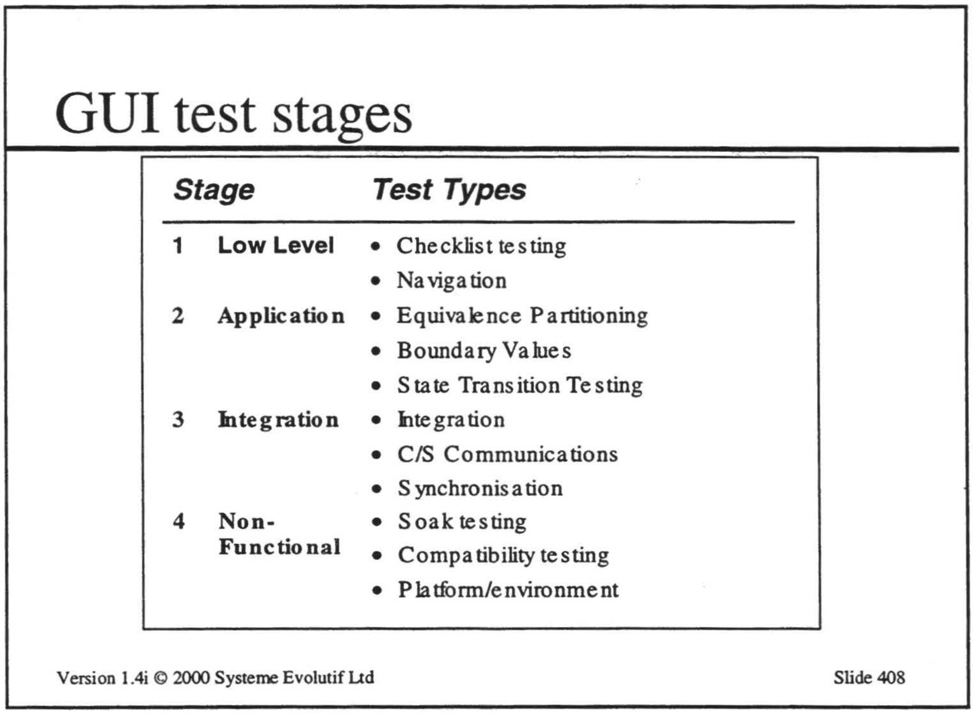
Slide 406



GUI testing

- GUI capture/replay tools are now much more sophisticated than character based
- Market leading tools:
  - are object-oriented
  - use visual development tool style
  - fully programmable
- Useful throughout the life-cycle.

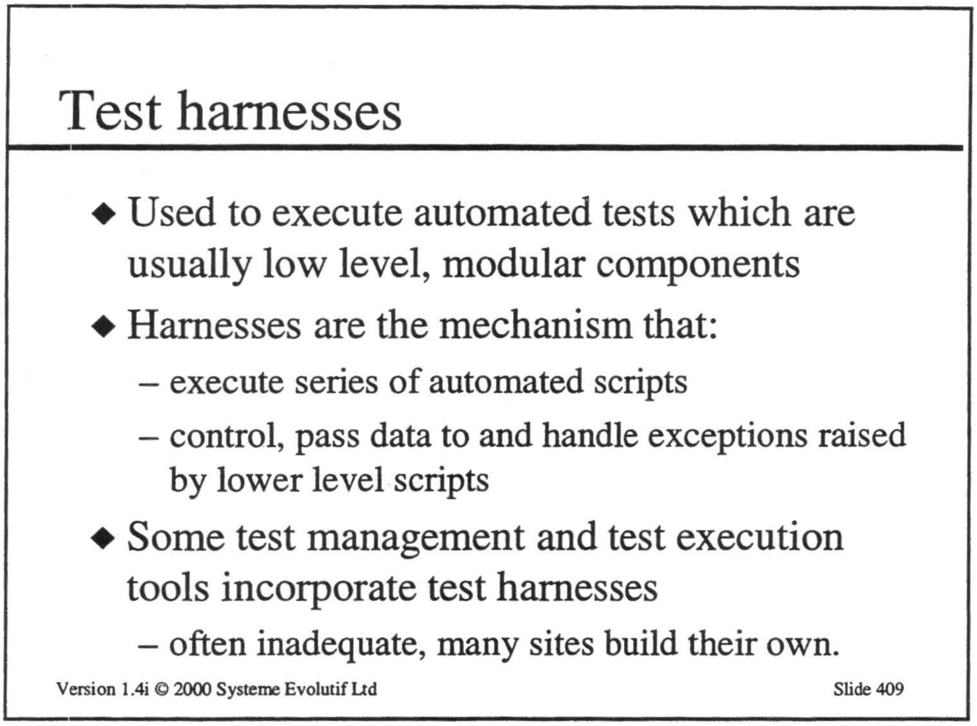
Version 1.4i © 2000 Systeme Evolutif Ltd　Slide 407



GUI test stages

	Stage	*Test Types*
1	**Low Level**	• Checklist testing • Navigation
2	**Application**	• Equivalence Partitioning • Boundary Values • State Transition Testing
3	**Integration**	• Integration • C/S Communications • Synchronisation
4	**Non-Functional**	• Soak testing • Compatibility testing • Platform/environment

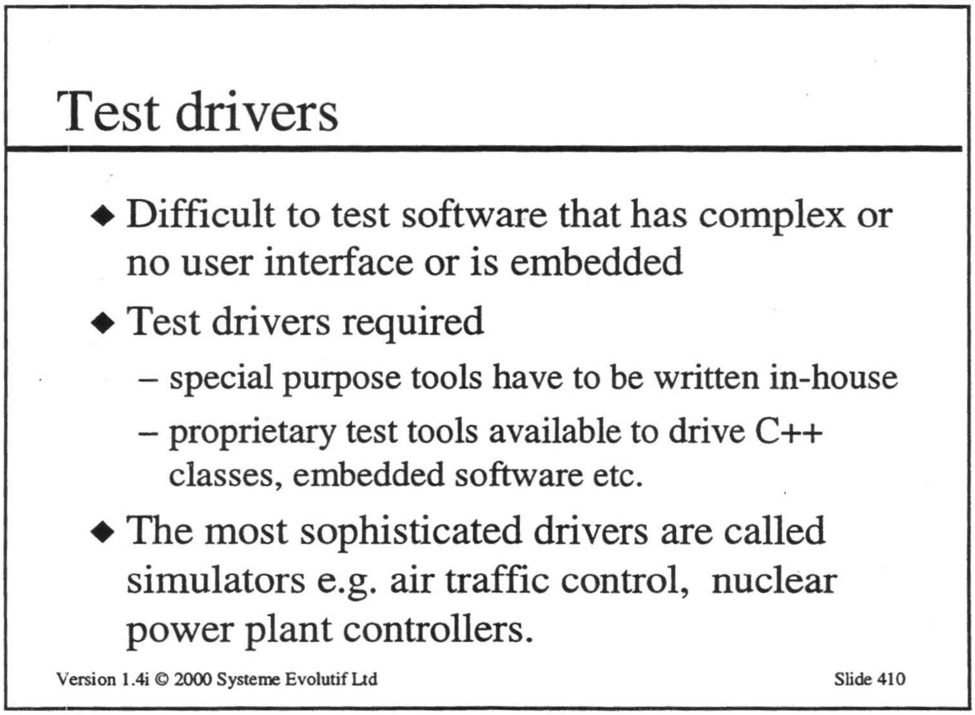
Version 1.4i © 2000 Systeme Evolutif Ltd　Slide 408



Test harnesses

- ◆ Used to execute automated tests which are usually low level, modular components
- ◆ Harnesses are the mechanism that:
  - – execute series of automated scripts
  - – control, pass data to and handle exceptions raised by lower level scripts
- ◆ Some test management and test execution tools incorporate test harnesses
  - – often inadequate, many sites build their own.

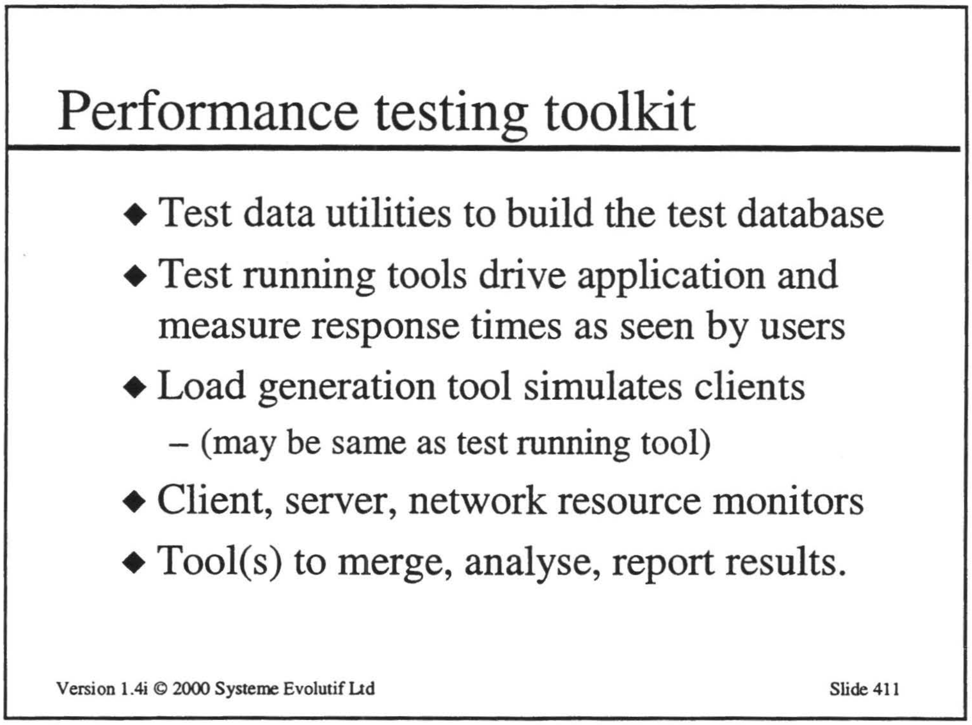
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 409



Test drivers

- ◆ Difficult to test software that has complex or no user interface or is embedded
- ◆ Test drivers required
  - – special purpose tools have to be written in-house
  - – proprietary test tools available to drive C++ classes, embedded software etc.
- ◆ The most sophisticated drivers are called simulators e.g. air traffic control, nuclear power plant controllers.

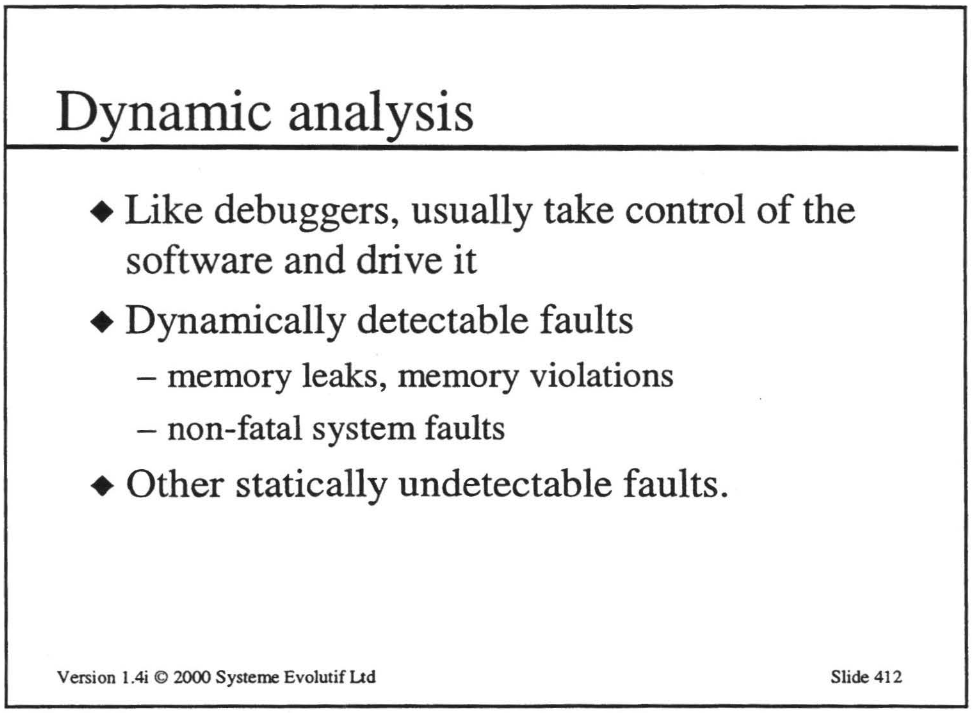
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 410



Performance testing toolkit

- ◆ Test data utilities to build the test database
- ◆ Test running tools drive application and measure response times as seen by users
- ◆ Load generation tool simulates clients
  - – (may be same as test running tool)
- ◆ Client, server, network resource monitors
- ◆ Tool(s) to merge, analyse, report results.

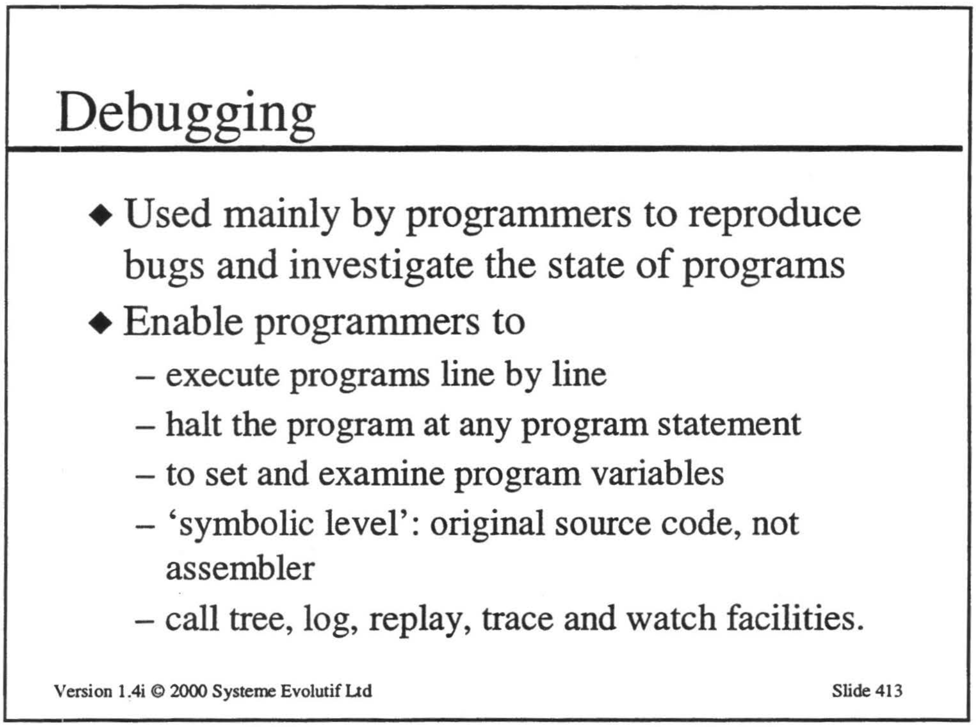
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 411



Dynamic analysis

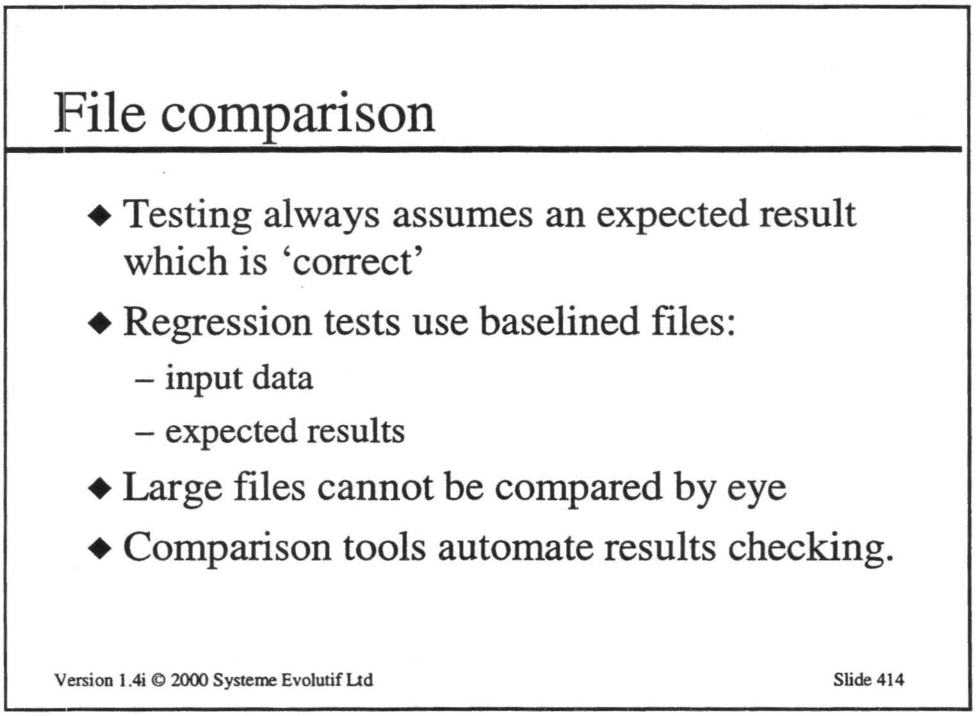
- ◆ Like debuggers, usually take control of the software and drive it
- ◆ Dynamically detectable faults
  - – memory leaks, memory violations
  - – non-fatal system faults
- ◆ Other statically undetectable faults.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 412



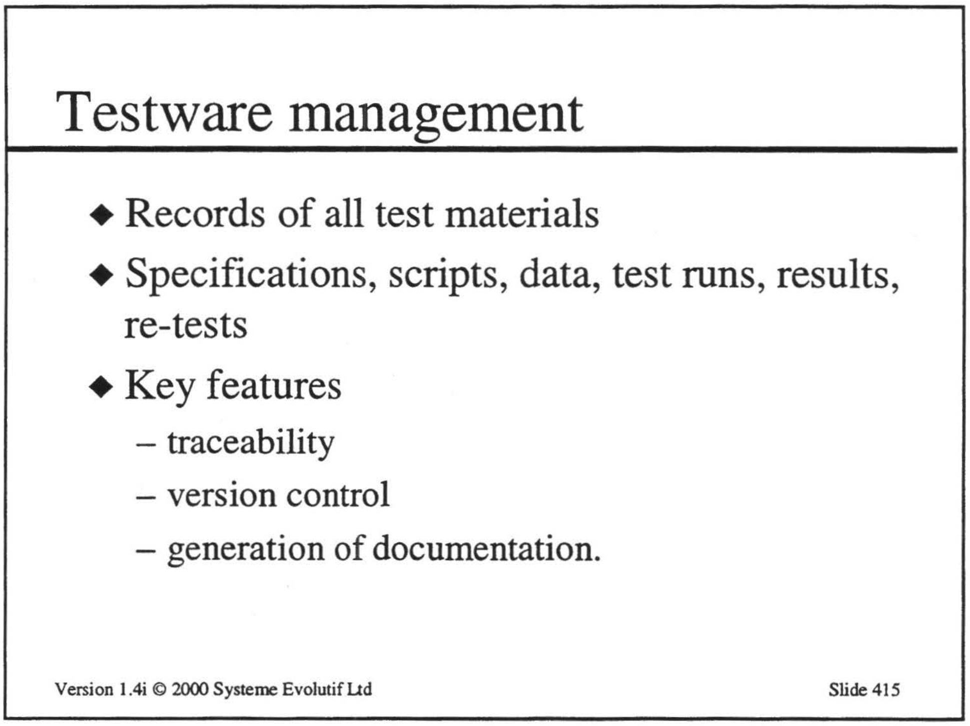
Debugging

- ◆ Used mainly by programmers to reproduce bugs and investigate the state of programs
- ◆ Enable programmers to
  - – execute programs line by line
  - – halt the program at any program statement
  - – to set and examine program variables
  - – 'symbolic level': original source code, not assembler
  - – call tree, log, replay, trace and watch facilities.



File comparison

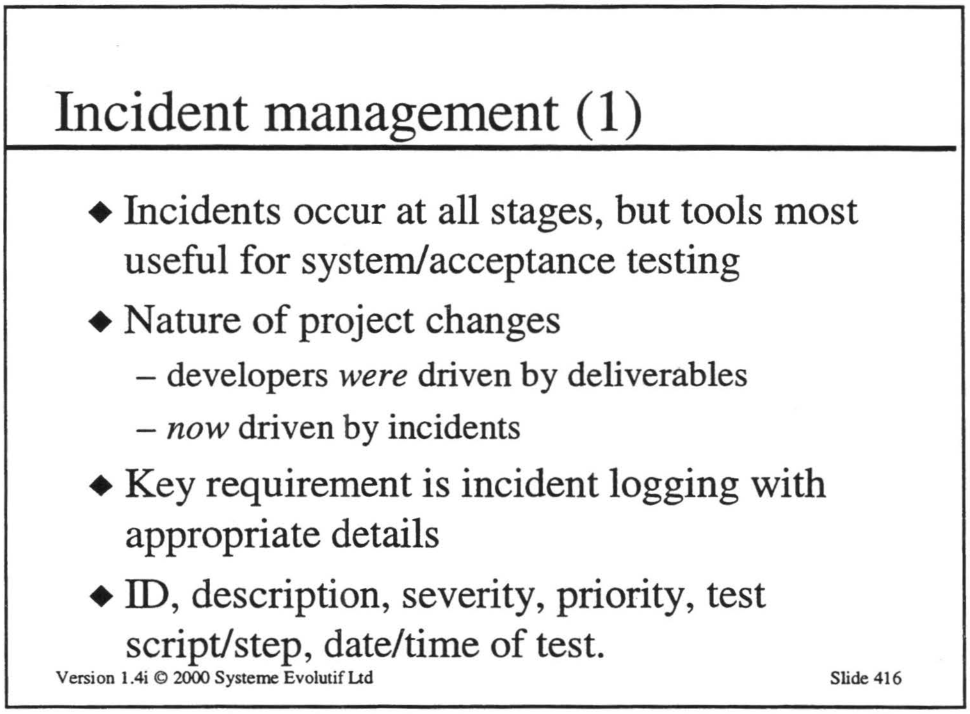
- ◆ Testing always assumes an expected result which is 'correct'
- ◆ Regression tests use baselined files:
  - – input data
  - – expected results
- ◆ Large files cannot be compared by eye
- ◆ Comparison tools automate results checking.



Testware management

- Records of all test materials
- Specifications, scripts, data, test runs, results, re-tests
- Key features
  - traceability
  - version control
  - generation of documentation.

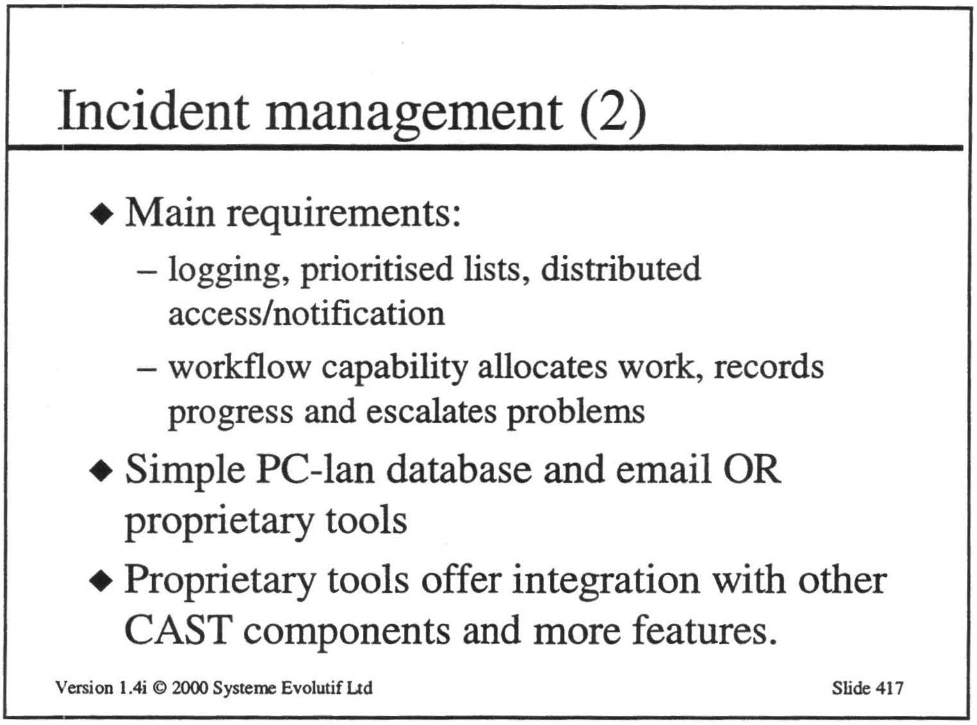
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 415



Incident management (1)

- Incidents occur at all stages, but tools most useful for system/acceptance testing
- Nature of project changes
  - developers *were* driven by deliverables
  - *now* driven by incidents
- Key requirement is incident logging with appropriate details
- ID, description, severity, priority, test script/step, date/time of test.

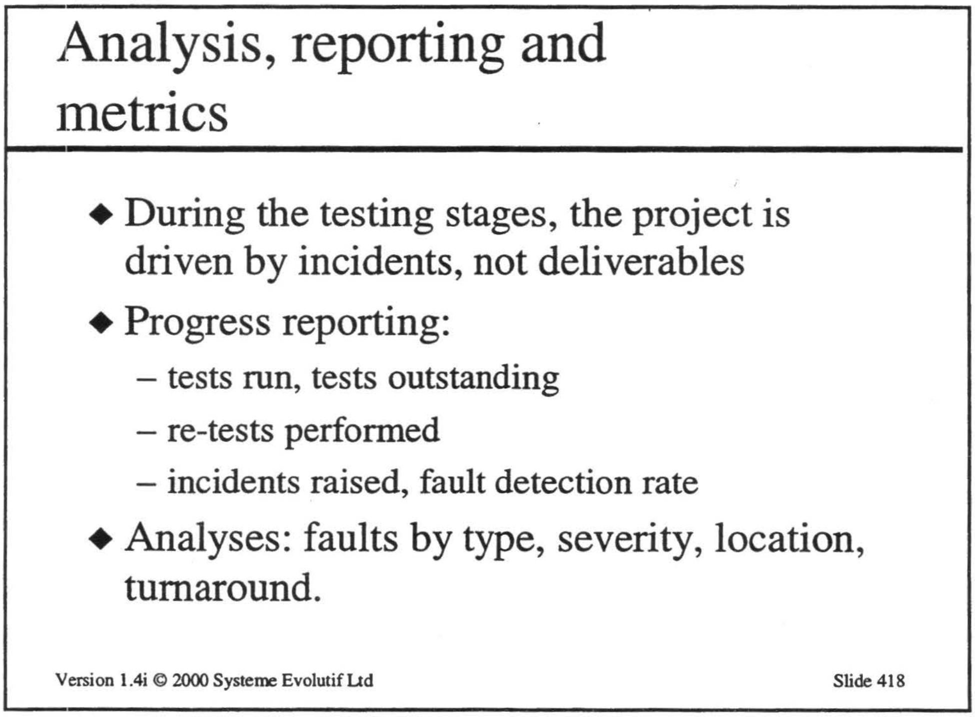
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 416



Incident management (2)

- Main requirements:
  - logging, prioritised lists, distributed access/notification
  - workflow capability allocates work, records progress and escalates problems
- Simple PC-lan database and email OR proprietary tools
- Proprietary tools offer integration with other CAST components and more features.

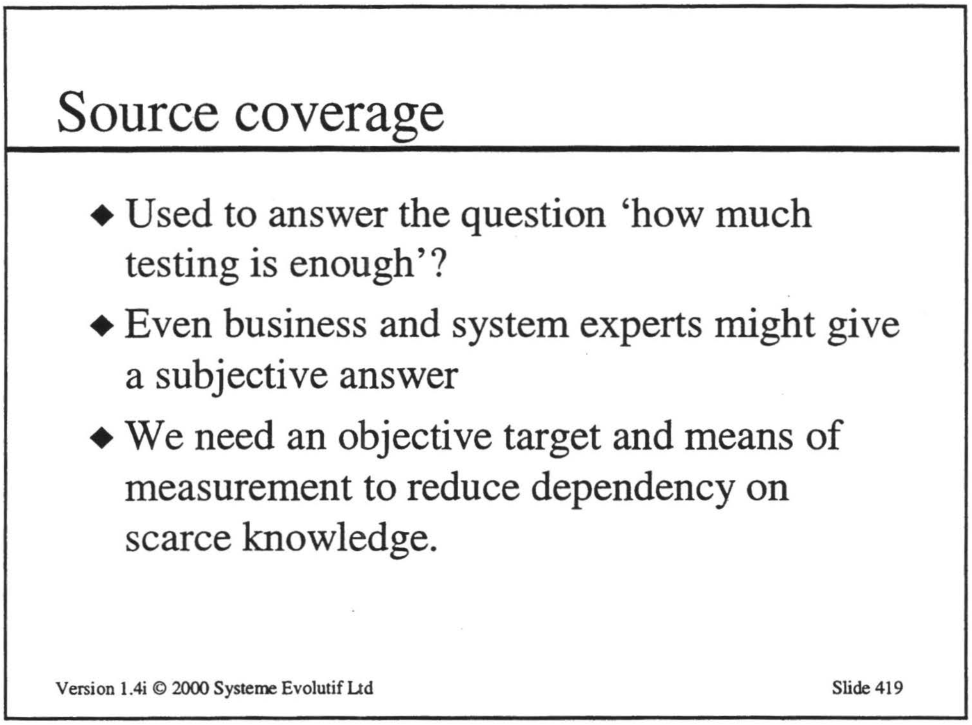
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 417



Analysis, reporting and metrics

- During the testing stages, the project is driven by incidents, not deliverables
- Progress reporting:
  - tests run, tests outstanding
  - re-tests performed
  - incidents raised, fault detection rate
- Analyses: faults by type, severity, location, turnaround.

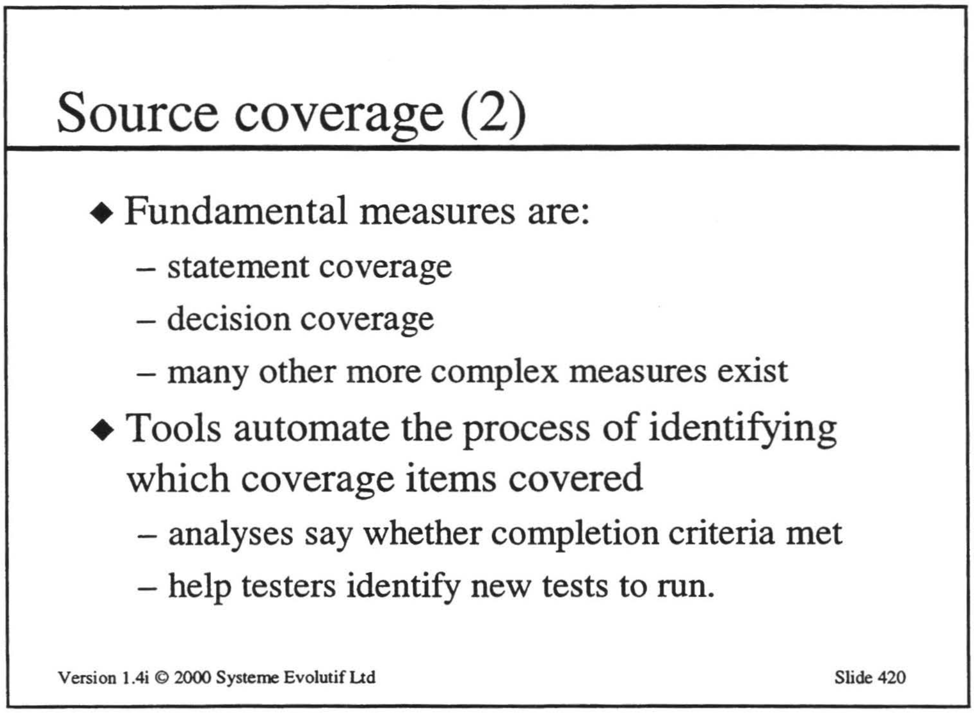
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 418



Source coverage

- ◆ Used to answer the question 'how much testing is enough'?
- ◆ Even business and system experts might give a subjective answer
- ◆ We need an objective target and means of measurement to reduce dependency on scarce knowledge.

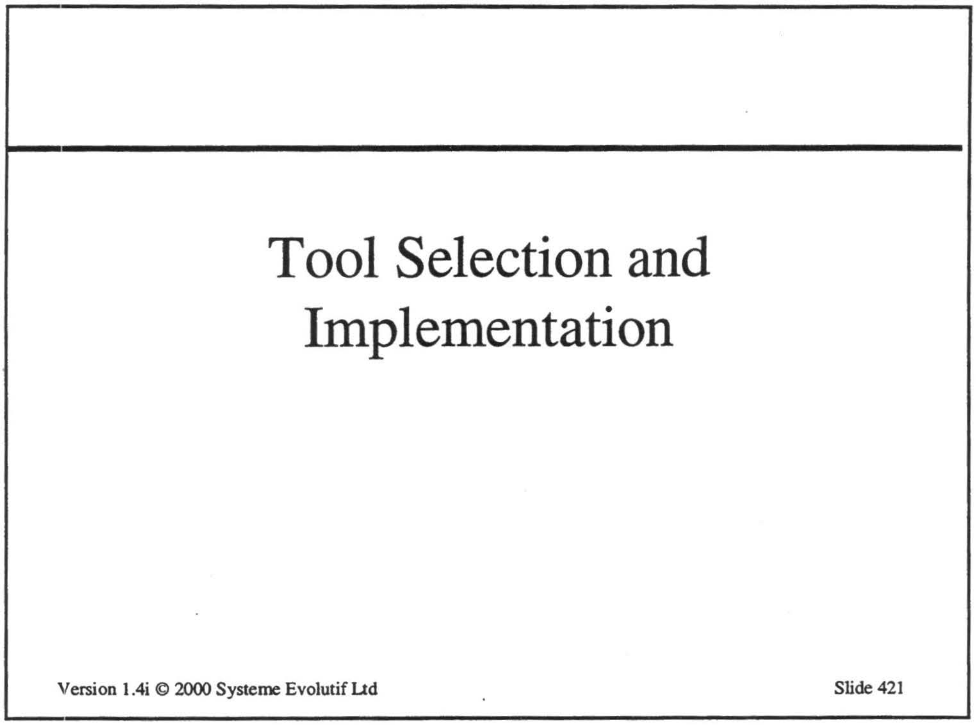
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 419



Source coverage (2)

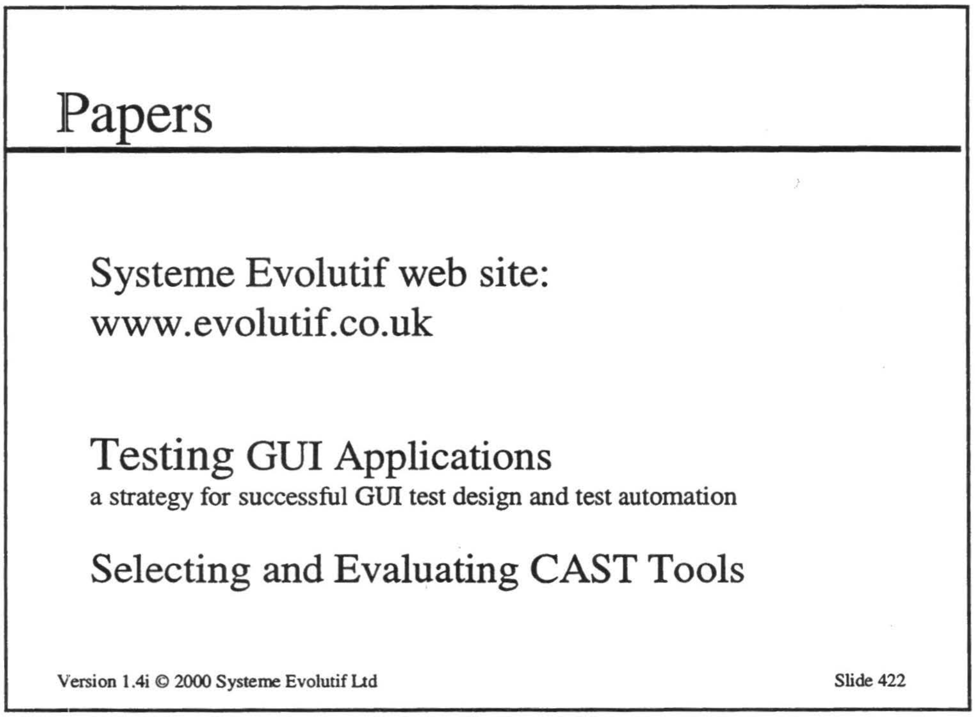
- ◆ Fundamental measures are:
  - – statement coverage
  - – decision coverage
  - – many other more complex measures exist
- ◆ Tools automate the process of identifying which coverage items covered
  - – analyses say whether completion criteria met
  - – help testers identify new tests to run.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 420



Tool Selection and Implementation

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 421



Papers

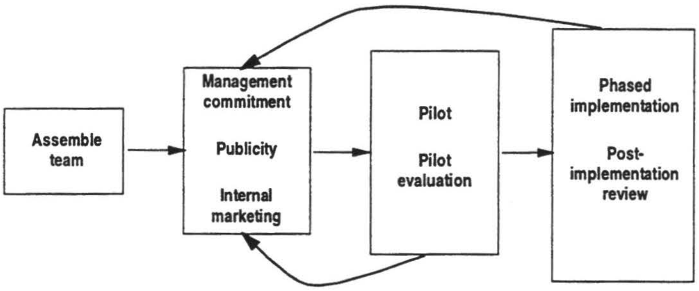
Systeme Evolutif web site:
www.evolutif.co.uk

Testing GUI Applications
a strategy for successful GUI test design and test automation

Selecting and Evaluating CAST Tools

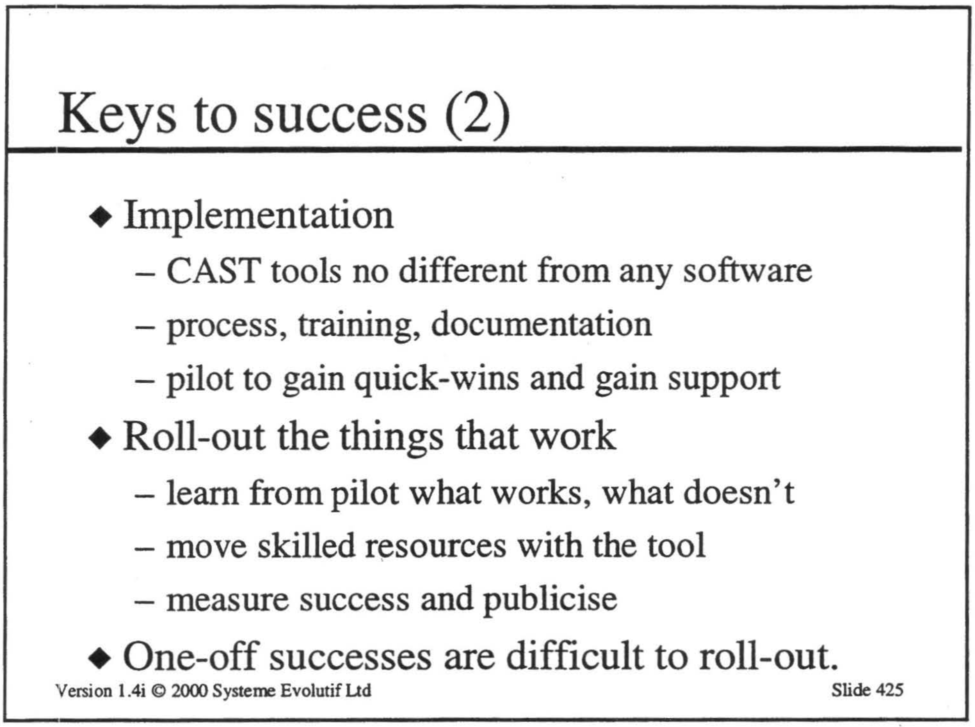
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 422

Tool implementation process



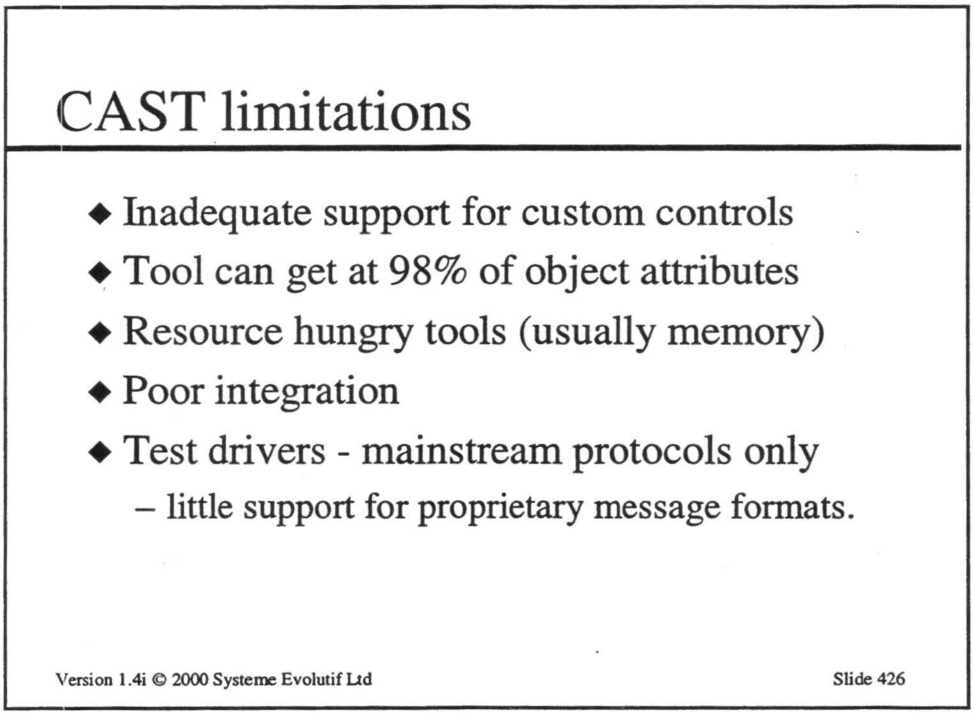
Keys to success

- Selling the concept
  - commitment to testing a pre-requisite
  - tools can save time/money but only if time is currently being spent on the task to be automated
- Selecting the right tool
  - tool should fit the test process or you will have to refine/develop test process at the same time
  - define the stages of testing the tool supports
  - not all testing can be automated!



Keys to success (2)

- Implementation
  - CAST tools no different from any software
  - process, training, documentation
  - pilot to gain quick-wins and gain support
- Roll-out the things that work
  - learn from pilot what works, what doesn't
  - move skilled resources with the tool
  - measure success and publicise
- One-off successes are difficult to roll-out.



CAST limitations

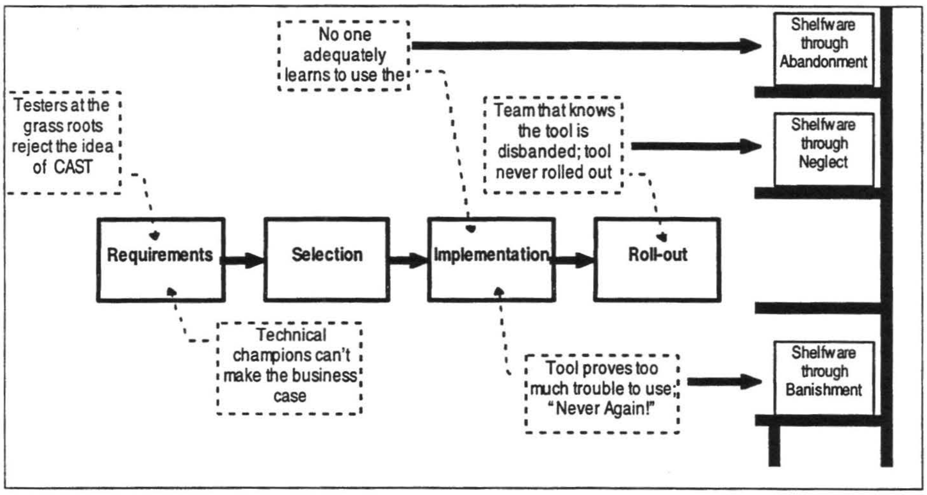
- Inadequate support for custom controls
- Tool can get at 98% of object attributes
- Resource hungry tools (usually memory)
- Poor integration
- Test drivers - mainstream protocols only
  - little support for proprietary message formats.

CAST availability

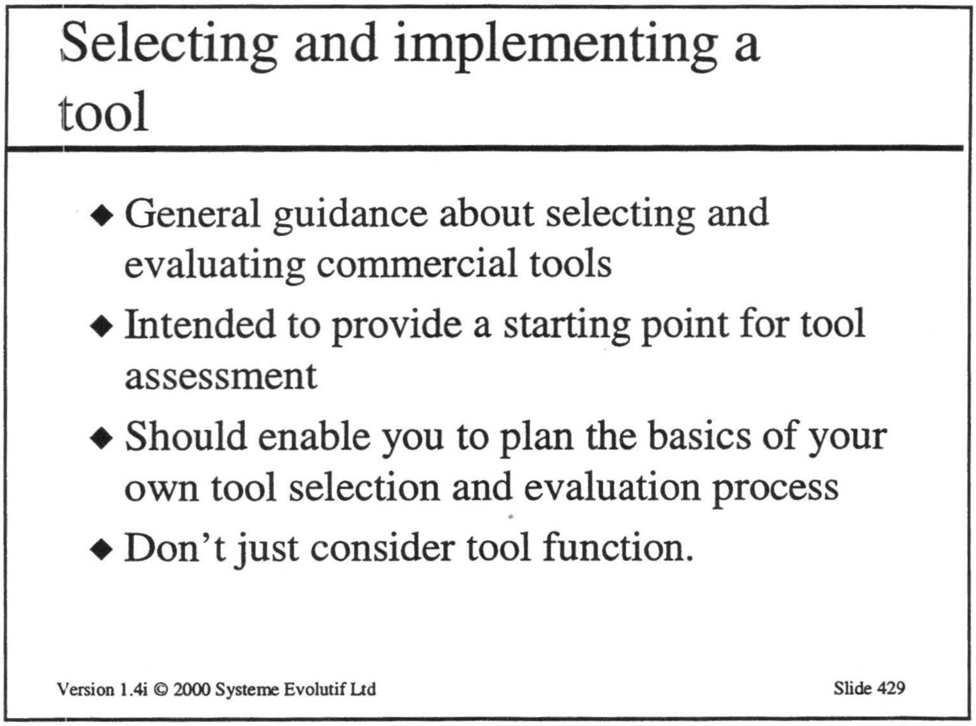
- ◆ Available on Unix but which?
- ◆ You use multiple platforms - but can't justify buying tools for every platform
- ◆ Available tools don't match your work practices
- ◆ Available tools are not integrated
- ◆ There are no tools available for your platform.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 427

Three routes to "shelfware"


No one adequately learns to use the
Shelfware through Abandonment
Testers at the grass roots reject the idea of CAST
Team that knows the tool is disbanded; tool never rolled out
Shelfware through Neglect
Requirements
Selection
Implementation
Roll-out
Technical champions can't make the business case
Tool proves too much trouble to use; "Never Again!"
Shelfware through Banishment

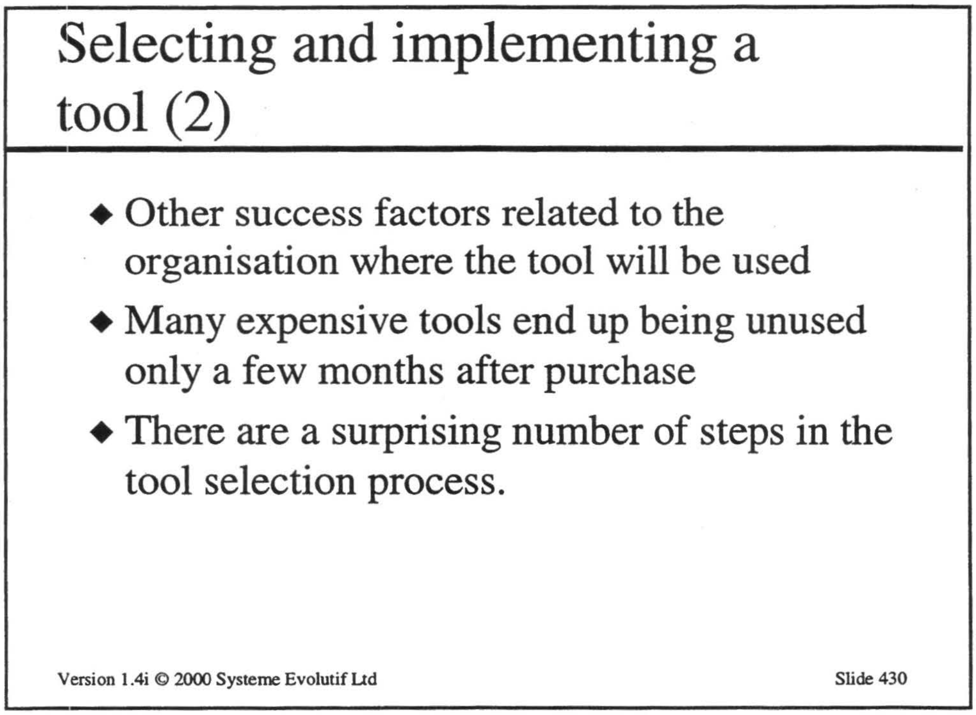
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 428



Selecting and implementing a tool

- ◆ General guidance about selecting and evaluating commercial tools
- ◆ Intended to provide a starting point for tool assessment
- ◆ Should enable you to plan the basics of your own tool selection and evaluation process
- ◆ Don't just consider tool function.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 429

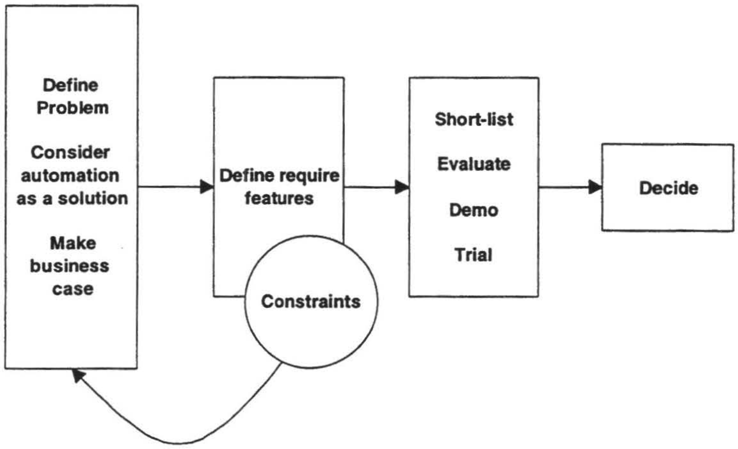


Selecting and implementing a tool (2)

- ◆ Other success factors related to the organisation where the tool will be used
- ◆ Many expensive tools end up being unused only a few months after purchase
- ◆ There are a surprising number of steps in the tool selection process.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 430

Overview of the selection process

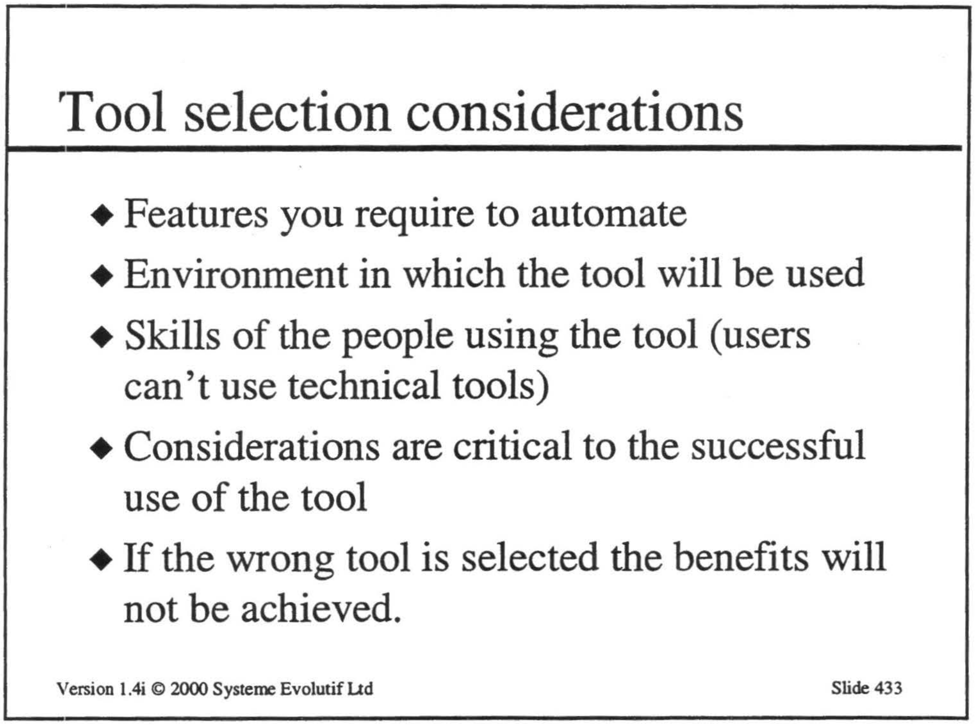


Version 1.4i © 2000 Systeme Evolutif Ltd Slide 431

Where to start

- You want to make your testing process more efficient through the use of a software testing tool
- Wide variety of tools available but which one to buy?

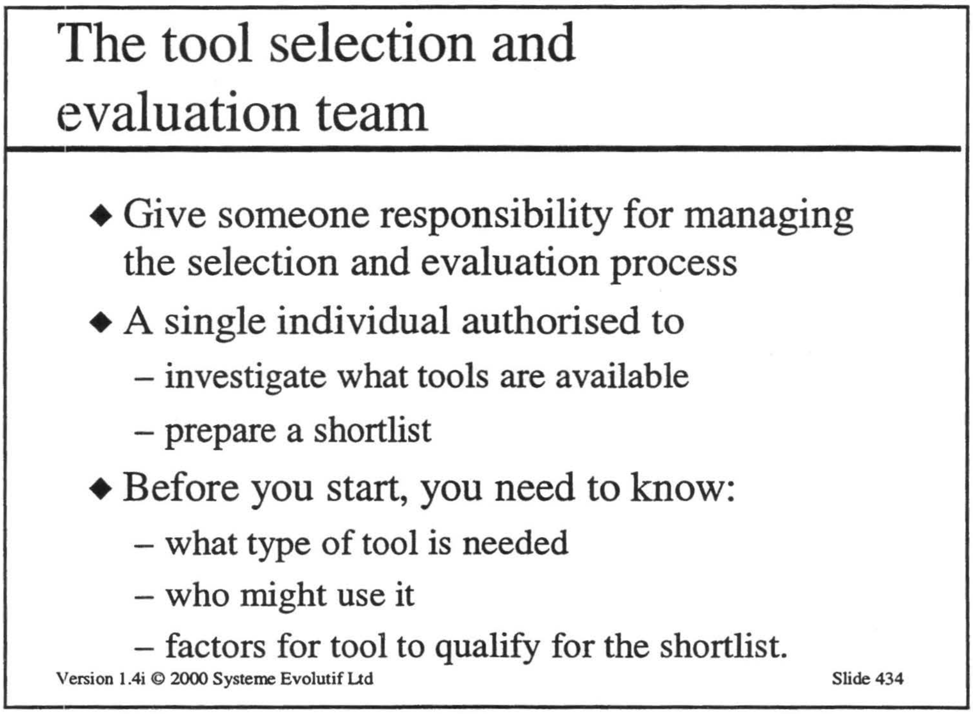
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 432



Tool selection considerations

- ◆ Features you require to automate
- ◆ Environment in which the tool will be used
- ◆ Skills of the people using the tool (users can't use technical tools)
- ◆ Considerations are critical to the successful use of the tool
- ◆ If the wrong tool is selected the benefits will not be achieved.

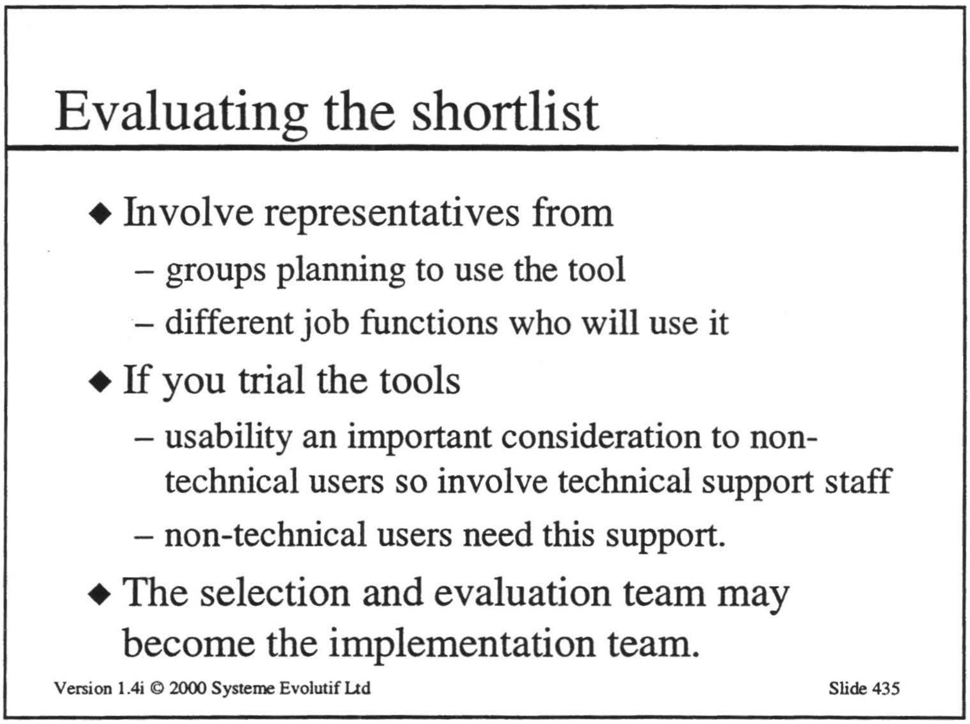
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 433



The tool selection and evaluation team

- ◆ Give someone responsibility for managing the selection and evaluation process
- ◆ A single individual authorised to
  - – investigate what tools are available
  - – prepare a shortlist
- ◆ Before you start, you need to know:
  - – what type of tool is needed
  - – who might use it
  - – factors for tool to qualify for the shortlist.

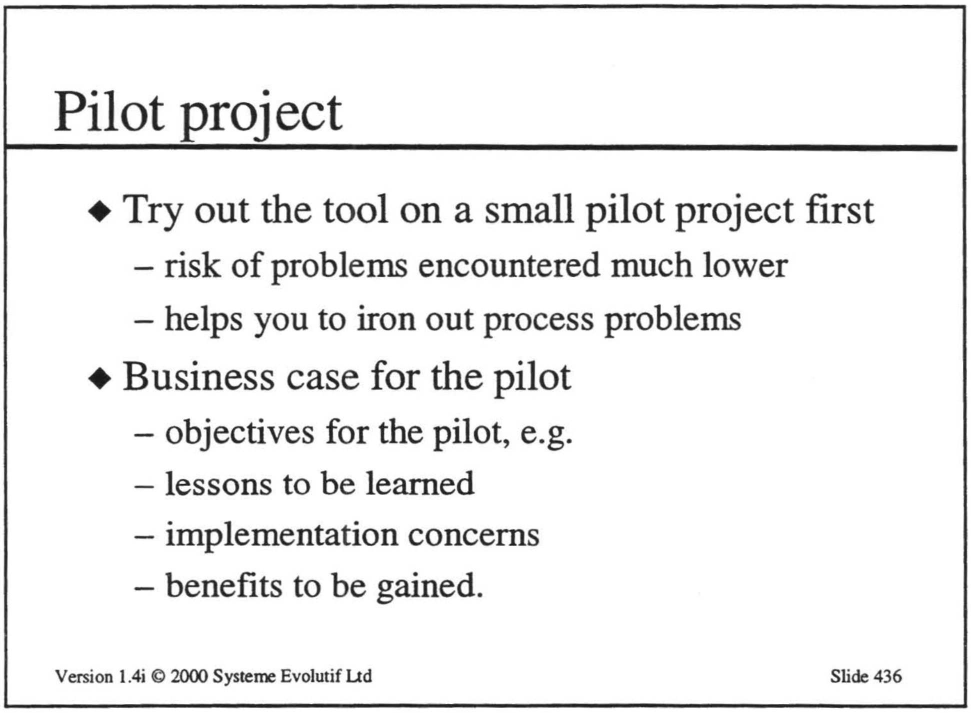
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 434



Evaluating the shortlist

- Involve representatives from
  - groups planning to use the tool
  - different job functions who will use it
- If you trial the tools
  - usability an important consideration to non-technical users so involve technical support staff
  - non-technical users need this support.
- The selection and evaluation team may become the implementation team.

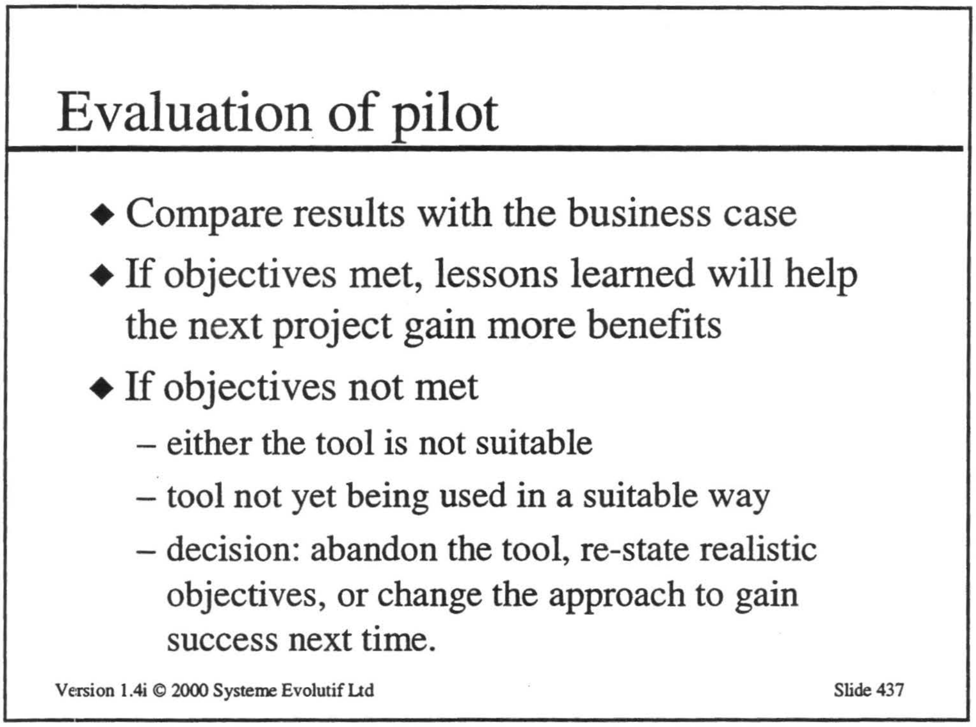
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 435



Pilot project

- Try out the tool on a small pilot project first
  - risk of problems encountered much lower
  - helps you to iron out process problems
- Business case for the pilot
  - objectives for the pilot, e.g.
  - lessons to be learned
  - implementation concerns
  - benefits to be gained.

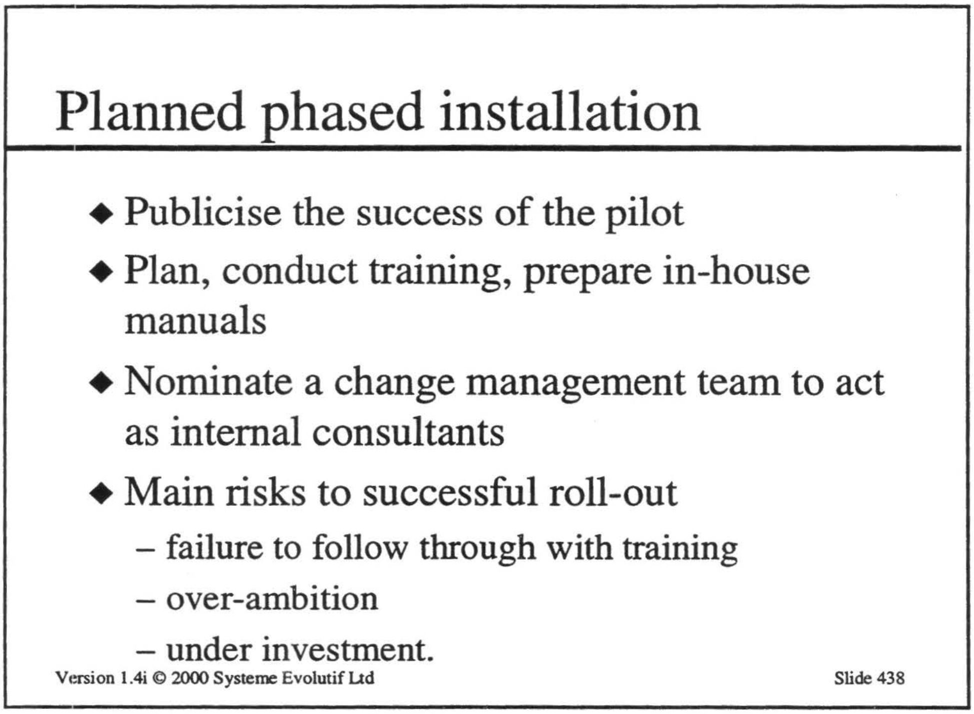
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 436



Evaluation of pilot

- Compare results with the business case
- If objectives met, lessons learned will help the next project gain more benefits
- If objectives not met
  - either the tool is not suitable
  - tool not yet being used in a suitable way
  - decision: abandon the tool, re-state realistic objectives, or change the approach to gain success next time.

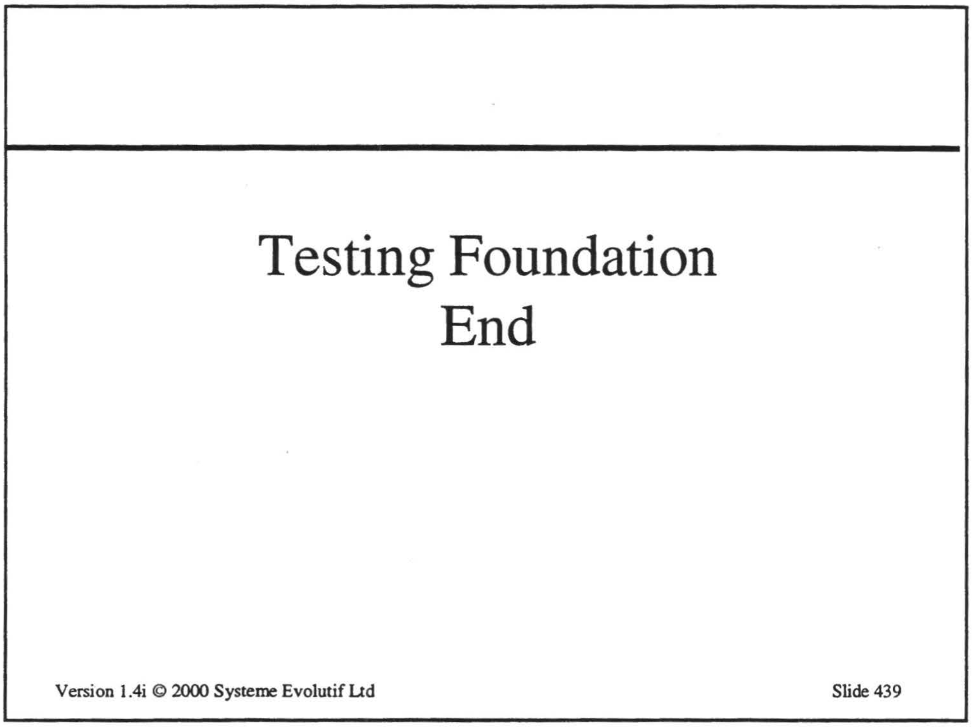
Version 1.4i © 2000 Systeme Evolutif Ltd Slide 437



Planned phased installation

- Publicise the success of the pilot
- Plan, conduct training, prepare in-house manuals
- Nominate a change management team to act as internal consultants
- Main risks to successful roll-out
  - failure to follow through with training
  - over-ambition
  - under investment.

Version 1.4i © 2000 Systeme Evolutif Ltd Slide 438



Testing Foundation End

Version 1.4i © 2000 Systeme Evolutif Ltd

Slide 439

1. EQUIVALENCE PARTITIONS AND BOUNDARY VALU E ANALYSIS

For each of the following examples:

- derive the valid and invalid equivalence partitions and
- the boundary values to be used.

1. An airliner can carry 180 economy passengers, and 44 club class. A marketing database uses a data entry screen to capture flight passenger statistics and it validates the number of economy and club passengers on each flight.
2. A system produces shift rosters for air traffic controllers the day before the shift. For every shift, radar positions must be filled by staff who performed the same role no more than ninety days previously. On the roster-input screen, when a controller is entered for a radar position, the system must check the ‘role last done’ is within the acceptable range.
3. A packet of biscuits must weigh 200g with a 2% tolerance if it is to pass a QC check and be fit for customer sale. A computerised weighing machine is used to perform the check. The machine will allow the packet to pass if the weight is within the 2% tolerance, but reject it if not.
4. When a taxable salary for a person is input to a program it works out a 'tax-due' figure and then displays a tax category. The tax rate changes from 25% to 35% at £30,000 salary. Persons paying up to £15,000 tax are categorised as 'low-bracket' and over £15,000 as 'high-bracket'.

2. PATH TESTING

In the example code below, what test cases are required to achieve:

- 100% statement testing
- 100% branch/decision testing
- NB: 'else' and 'endif' statements do not count as executable statements - disregard them in the calculation of coverage.

Example 1

	Test Cases	
	Statement Coverage	Branch Coverage

If x = 23 then
      DoThis
      DoThat
      DoOther
Endif		

Example 2

	Test Cases	
	Statement Coverage	Branch Coverage

If p = true then
      DoThis
      DoThat
Else
      DoOther
Endif		

Example 3

	Test Cases	
	Statement Coverage	Branch Coverage

If p = true then
     If x = 23 then
           DoThis
           DoThat
     Endif
Else
     DoOther
Endif		

Example 4

	Test Cases	
	Statement Coverage	Branch Coverage

If p = true then
     If x = 23 then
           DoThis
           DoThat
     Else
           Doother
     Endif
Else
     DoLittle
Endif		

3. REVIEW EXERCISE

The document to be reviewed is an extract from a draft guide for managing, maintaining and executing regression tests. Read the document and make comments in the form below. Look for issues of clarity, consistency, completeness and correctness. Do not search for spelling, punctuation or grammatical faults, but log them if you come across them. Use the form below to record your observations.

Review of: Regression Test Guidelines **Version:**

Reviewer: **Date:**

No.	**Location**	**Description**	**Fault Type:** **e.g. clarity, consistency etc.**

Review of: Regression Test Guidelines		**Version:**	
Reviewer:	**Date:**		
No.	**Location**	**Description**	**Fault Type:** **e.g. clarity,** **consistency** **etc.**

DETERMINING WHICH TEST PROCEDURES TO IN CLUDE IN EACH REGRESSION TEST LIBRARY

2.1 Regression Testing Policy

Each project will document its exit criteria and its approach to (or policy on) regression testing for each stage of testing in the project's Master Test Plan (MTP). This will define whether the regression test policy is time constrained (e.g. projects which must be implemented by 6th April) or whether the implementation will be delayed if exit criteria are not met (e.g. if all regression tests have not passed).

The regression testing approach will also specify the time slot allowed for regression testing which in turn will determine which RTL will be used. The greater the time slot available, the greater the number of regression Test Procedures that can be run.

If the regression test will be time constrained (as it currently is for Live User Testing (LUT)), then the duration of the regression test should be specified at the earliest opportunity.

2.2 Allocating Test Procedures to Regression Test Libraries

The overriding objective in determining which Test Procedures to include in one or more Regression Test Libraries (RTLs) is to minimise the time needed for regression testing, without sacrificing the assurance that new faults have not been introduced into existing software components.

In other words the goal for an RTL which must be able to run in only one hour is that it must include only the best set of Test Procedures capable of being run in that time.

From this it follows that all of the Test Procedures included in a one hour RTL will also be included in a three hour RTL, as well as the set of next best Test Procedures.

Selection Criteria

The achievement of this objective of including only the best set of Test Procedures in the relevant RTL involves prioritisation. Test Procedures which meet the criteria listed below, should be selected as contenders to be included in one or more of the RTLs.

- Test Procedures which exercise critical functionality
- Test Procedures which cover error-prone parts of the system
- Test Procedures which use boundary values rather than mid-range values
- Test Procedures which do not have much overlap with any other Test Procedure in the RTL
- Test Procedures which have not been designed to detect the same fault as any other Test Procedure in the RTL.

Additional Selection Criteria

If the regression testing is to be performed manually, it is likely that the initial set of Test Procedures which meets the above criteria will need to be further reduced in order for a regression test to complete in its allotted time slot.

If the initial set of Test Procedures needs to be reduced further, before being included in one or more RTLs, then the following additional criteria can be applied to each remaining Test Procedure to highlight the best to include as regression tests:

- Test Procedures which maximise the number of test conditions per test step
- Test Procedures which are quickest to run (usually the automated ones)

- Test Procedures to be most easily automated
- Test Procedures which are independent, that is they can be run in any order without needing other Test Procedures to be run.

2.3 Tailoring Regression Test Libraries for Specific Releases

One way of maximising test coverage in a time restricted RTL is to combine Test Procedures.

It will often be possible to combine a number of Test Procedures into a single Test Procedure thereby running only one Test Procedure to cover all of the test conditions addressed by a number of Test Procedures. This is the best way to achieve the criterion listed earlier of maximising the number of test conditions per test step.

Whenever Test Procedures are combined into a single Test Procedure, a new Test Procedure identifier (with an 'R' prefix)must be assigned to it. The Test Analyst must also record a mapping of the identifiers of the original Test Procedures with the identifier of the new replacement Test Procedure, in order to preserve traceability back to the Business Process or the Behaviour.

As well as combining Test Procedures, Test Analysts should rotate the Test Procedures which are included in each RTL (e.g. substituting 'Buy' Test Procedures with 'Sell' Test Procedures and vice versa on alternate test runs).

Similarly, provided time permits, Test Analysts should vary the data values contained within the set of Test Procedures. However time constraints may preclude this if regression testing is manual.

SELECTING WHICH REGRESSION TEST PROCEDU RES TO EXECUTE

3.1 Tailoring Regression Test Libraries for Specific Releases

The purpose of regression testing is to test parts of the system which are not expected to have been impacted by the additions or changes to the system under test. From this it follows that each RTL should cover a wide range of the critical system functions to provide confidence that unexpected side effects have not been introduced with the changes. The Test Procedures allocated to a particular RTL are allocated without regard to a specific software release.

It will therefore be beneficial, whenever time permits, to select additional Test Procedures to be run after running all the Test Procedures in the appropriate RTL. The additional Test Procedures should be existing Test Procedures which were designed to test areas of the system which are inter-related with the new or changed software components.

In addition to including Test Procedures which exercise the software most closely related /linked to changed or new software components, it is also worth including Test Procedures which uncovered faults the last time they were run as additional Test Procedures.

In this way it is possible, given very short notice of a need to run a four hour regression test, to select the best Test Procedures, by running the set of Test Procedures that comprise the three hour RTL, complemented by including the set of Test Procedures which can be run in an hour and which are most closely related to the changed software component. This is achievable because the analysis of selecting the best Test Procedures across the breadth of the system has already taken place, while the Function / Feature Tested section of the Test Procedure Inventory can be searched to locate the Test Procedures most likely to be impacted by the changes.

TEST PROCEDURE INVENTORY				***Regression Test Library Type***			
Procedure ID	***Function / Feature Tested***	***Auto-mated / Manual***	***Est. Run Time (mins)***	***One Hour RTL Test***	***Three Hour RTL Test***	***One Day RTL Test***	***Two Day RTL Test***
SFOA001	CSE finds client details, **accesses corporate library view**, checks price, **writes to scratch pad** and **sends an internal e-mail**	M	8		Y	Y	Y
SFOA002	CSE finds client details, accesses corporate library view, checks price, client places a **buy order** (fixed quantity of securities), a **dealer views the order** on the pending queue, the order is **dealt** and the order is processed by the AS/400 batch maintenance	A	1	Y	Y	Y	Y
SFOA003	CSE finds client details, accesses corporate library view, checks price, client places a **sell order** (fixed quantity of securities), a **dealer views the order** on the pending queue, the order is **dealt** and the order is processed by the AS/400 batch maintenance	M	10		Y	Y	Y
SFOA004	CSE finds client details, accesses corporate library view, checks price, client places an **invest order** (buy fixed monetary amount), a **dealer views the order** on the pending queue, the order is **dealt** and the order is processed by the AS/400 batch maintenance	M	12				
SFOA005	CSE finds client details, accesses corporate library view, checks price, client places a **realise order** (sell fixed monetary amount), a **dealer views the order** on the pending queue, the order is **dealt** and the order is processed by the AS/400 batch maintenance	A	1	Y	Y	Y	Y
UFOA001	CSE finds client details, accesses corporate library view, checks price, client places a **buy unachievable limit order** (fixed quantity of securities at a fixed price), it is sent to the **limit held queue**, the **limit expires** and is **rejected**	M	5				Y
UFOA002	CSE finds client details, accesses corporate library view, checks price, client places a **sell unachievable limit order** (fixed quantity of securities at a fixed price), it is sent to the **limit held queue**, the **limit expires** and is **rejected**	A	1	Y	Y	Y	Y

TEST PROCEDURE INVENTORY				Regression Test Library Type			
Procedure ID	Function / Feature Tested	Auto-mated / Manual	Est. Run Time (mins)	One Hour RTL Test	Three Hour RTL Test	One Day RTL Test	Two Day RTL Test
UFOA003	CSE finds client details, accesses corporate library view, checks price, client places an **invest unachievable limit order** (buy fixed monetary amount at a fixed price), it is sent to the **limit held queue**, the **limit expires** and is **rejected**	A	1	Y	Y	Y	Y
UFOA004	CSE finds client details, accesses corporate library view, checks price, client places a **realise unachievable limit order** (sell fixed monetary amount), it is sent to the **limit held queue**, the **limit expires** and is **rejected**	M	10				
UFOA005	CSE finds client details, accesses corporate library view, checks price, client places a **buy achievable limit** (fixed quantity of securities at a fixed price), the **MML criteria are not met**, a **dealer views the order** on the pending queue, the order is **dealt** and processed by the AS/400 batch maintenance	M	8				
UFOA006	CSE finds client details, accesses corporate library view, checks price, client places a **sell achievable limit order** (fixed quantity of securities at a fixed price), the **MML criteria are not met,** a **dealer views the order** on the pending queue, the **price is achieved** and the **order is dealt** and processed by the AS/400 batch maintenance	M	12				Y
UFOA007	CSE finds client details, accesses corporate library view, checks price, client places an **invest achievable limit order** (buy fixed monetary amount at a fixed price), the **MML criteria are not met,** a **dealer views the order** on the pending queue, the **price is achieved** and the order is **dealt** and processed by the AS/400 batch maintenance	M	10				Y
UFOA008	CSE finds client details, accesses corporate library view, checks price, client places a **realise achievable limit order** (sell fixed monetary amount), the **MML criteria are not met**, a **dealer views the order** on the pending queue, the **price is achieved** and the **order is dealt** and processed by the AS/400 batch maintenance	M	8				

*** End of document. ***

4. STOPPING TESTING

For each of the situations below, what action should be taken? Should you stop testing, continue testing or take some other action?

4.1. Situation 1

Scheduled end date:	10/2/94	Date today:	15/2/94
Number of scripts in test plan:	17	Scripts run:	17
High severity errors reported:	4	Open:	1
Medium severity errors reported:	11	Open:	3
Low severity errors reported:	3	Open:	0

4.2. Situation 2

Project phase start date:	1/12/93	Project phased planned end date:	31/3/94
Test planned start:	1/3/94	Test planned end date:	28/3/94
Actual test start date:	15/3/94	Current planned test end date:	28/3/94
Today's date:	28/3/94		
Number of scripts in test plan:	31	Scripts run: 7	
High severity errors reported:	0	Open: 0	
Medium severity errors reported:	5	Open: 0	
Low severity errors reported:	7	Open: 7	

4.3. Situation 3

Project phase start date:	1/12/93	Project phased planned end date:	31/3/94
Test planned start:	1/3/94	Test planned end date:	28/3/94
Actual test start date:	15/3/94	Current planned test end date:	28/3/94
Today's date:	28/3/94		
Number of scripts in test plan:	31	Scripts run: 27	
High severity errors reported:	2	Open: 0	
Medium severity errors reported:	46	Open: 11	
Low severity errors reported:	18	Open: 7	

4.4. Situation 4

Error counts for the last six weeks of testing:

Week	1	2	3	4	5	6
Errors	4	17	16	7	5	3

5. EQUIVALENCE PARTITIONS AND BOUNDARY VALU E ANALYSIS ANSWERS

Equivalence Partitioning Exercise - Answers

1. Valid: 0-180 economy and 0-44 club

   Invalid: less than zero economy, more than 180 economy, less than 0 club, more than 44 club.

   BV: -1, 0, 1, 179, 180, 181 economy, -1, 0, 1, 43, 44, 45 club

2. Valid: date between 1 and 90 days previous to shift date

   Invalid: date more than 90 days previous, or less than 1 day previous.

   BV: 91, 90, 89, 1, 0, -1 days previous

3. Valid: 196-204 grams

   Invalid: <196 grams and > 204 grams

   BV: 195.9, 196.0, 196.1, 203.9, 204.0, 204.1 grams

4. Input Equivalence Classes:

   Valid classes:

   salary £0.00-29,999.99

   salary >= £30,000

   Invalid class:

   salary <£0.00

   Output Equivalence Classes:

   Valid:

   tax payable between £0.00 and £15,000

   tax payable >£15,000

   Invalid:

   tax payable <£0.00

   Input BV:

   £-0.01, £0.00, £0.01, £29,999.99, £30,000.00, £30,000.01

   Output BV:

   £51,428.57, £51,428.58, £51,428.59

6. PATH TESTING - ANSWERS

Example 1

	Test Cases	
	Statement Coverage	Branch Coverage

If x = 23 then
      DoThis
      DoThat
      DoOther
Endif
	1. X = 23	1. X = 23
		2. X <> 23

Example 2

	Test Cases	
	Statement Coverage	Branch Coverage

If P = true then
      DoThis
      DoThat
else
      DoOther
Endif
	1. P = true
	2. P = false
		1. P = true
		2. P = false

Example 3

	Test Cases	
	Statement Coverage	Branch Coverage

If p = true then
     If x = 23 then
           DoThis
           DoThat
     Endif
Else
     DoOther
Endif
	1. P = true, x = 23
	2. p = false
		1. p = true, x=23
		2. p = true, x<>23
		3. p = false

Example 4

	Test Cases	
	Statement Coverage	Branch Coverage

If p = true then
     If x = 23 then
           DoThis
           DoThat
     Else
           Doother
     Endif
Else
     DoOther
Endif
	1. P = true, x = 23
	2. P = true, x<>23
	3. p = false
		1. p = true, x=23
		2. p = true, x<>23
		3. p = false

7. DETERMINING WHICH TEST PROCEDURES TO INC LUDE IN EACH REGRESSION TEST LIBRARY

2.1 Regression Testing Policy

Each project will document its exit criteria and its approach to (or policy on) regression testing for each stage of testing in the project's Master Test Plan (MTP).

Regression test policy covers two activities:

- Selection of tests to be included in the RTL
- Selection of tests to be executed.

The regression test policy may be constrained by time available to build regression tests (first bullet) or execute regression tests (second bullet).

Live User Testing (LUT) is usually time constrained so the time available for the execution of the regression test should be specified at the earliest opportunity.

2.2 Selecting Test Procedures for Inclusion in the RTL

The criteria below should be used to identify the Test Procedures to be included in the RTL.

Generic RTL Inclusion Criteria for a Test Procedure

- exercises critical functionality
- covers error-prone parts of the system
- exercises boundary values rather than 'mid-range' values
- is 'unique' i.e. it does not overlap with any other Test Procedure in the RTL
- addresses a specific fault or risk
- is automated, already, or can be easily automated
- exercises an above average number of test conditions
- can be run independently, i.e. there are not run-time dependencies on other procedures.

When Test Procedures are included in the RTL, for the purposes of automation, it is often easier to merge similar tests to create more efficient test that can benefit from iterative execution that tools can easily be made to do. These tests are, in effect new tests.

2.3 Selecting RTL Test Procedures to Execute

The criteria below should be used to identify Test Procedures to be included in the RTL.

Generic RTL Execution Criteria for a Test Procedure

- exercises the most critical functionality that has changed
- exercises functionality believed to be at most risk of regression faults (in the vicinity of changes)
- automated
- manual but can be run quickly.

In addition to including Test Procedures which exercise the software most closely related /linked to changed or new software components, it is also worth including Test Procedures which uncovered faults the last time they were run as additional Test Procedures.

The tests to be run in a 1 hour timeslot should be the tests that most closely meet the above criteria.

TEST PLAN OUTLINE (IEEE 829 FORMAT)

1) Test Plan Identifier
2) References
3) Introduction
4) Test Items
5) Software Risk Issues
6) Features to be Tested
7) Features not to be Tested
8) Approach
9) Item Pass/Fail Criteria
10) Suspension Criteria and Resumption Requirements
11) Test Deliverables
12) Remaining Test Tasks
13) Environmental Needs
14) Staffing and Training Needs
15) Responsibilities
16) Schedule
17) Planning Risks and Contingencies
18) Approvals
19) Glossary

IEEE TEST PLAN TEMPLATE

1 TEST PLAN IDENTIFIER

Some type of unique company generated number to identify this test plan, its level and the level of software that it is related to. Preferably the test plan level will be the same as the related software level. The number may also identify whether the test plan is a Master plan, a Level plan, an integration plan or whichever plan level it represents. This is to assist in coordinating software and testware versions within configuration management.

Keep in mind that test plans are like other software documentation, they are dynamic in nature and must be kept up to date. Therefore, they will have revision numbers.

You may want to include author and contact information including the revision history information as part of either the identifier section of as part of the introduction.

2 REFERENCES

List all documents that support this test plan. Refer to the actual version/release number of the document as stored in the configuration management system. Do not duplicate the text from other documents as this will reduce the viability of this document and increase the maintenance effort. Documents that can be referenced include:

- Project Plan
- Requirements specifications
- High Level design document
- Detail design document
- Development and Test process standards
- Methodology guidelines and examples
- Corporate standards and guidelines

3 INTRODUCTION

State the purpose of the Plan, possibly identifying the level of the plan (master etc.). This is essentially the executive summary part of the plan.

You may want to include any references to other plans, documents or items that contain information relevant to this project/process. If preferable, you can create a references section to contain all reference documents.

Identify the Scope of the plan in relation to the Software Project plan that it relates to. Other items may include, resource and budget constraints, scope of the testing effort, how testing relates to other evaluation activities (Analysis & Reviews), and possible the process to be used for change control and communication and coordination of key activities.

As this is the "Executive Summary" keep information brief and to the point.

4 TEST ITEMS (FUNCTIONS)

These are things you intend to test within the scope of this test plan. Essentially, something you will test, a list of what is to be tested. This can be developed from the software application inventories as well as other sources of documentation and information.

This can be controlled and defined by your local Configuration Management (CM) process if you have one. This information includes version numbers, configuration requirements where needed, (especially if multiple versions of the product are supported). It may also include key delivery schedule issues for critical elements.

Remember, what you are testing is what you intend to deliver to the Client.

This section can be oriented to the level of the test plan. For higher levels it may be by application or functional area, for lower levels it may be by program, unit, module or build.

5 SOFTWARE RISK ISSUES

Identify what software is to be tested and what the critical areas are, such as:

A. Delivery of a third party product.

B. New version of interfacing software

C. Ability to use and understand a new package/tool, etc.

D. Extremely complex functions

E. Modifications to components with a past history of failure

F. Poorly documented modules or change requests

There are some inherent software risks such as complexity; these need to be identified.

A. Safety

B. Multiple interfaces

C. Impacts on Client

D. Government regulations and rules

Another key area of risk is a misunderstanding of the original requirements. This can occur at the management, user and developer levels. Be aware of vague or unclear requirements and requirements that cannot be tested.

The past history of defects (bugs) discovered during Unit testing will help identify potential areas within the software that are risky. If the unit testing discovered a large number of defects or a tendency towards defects in a particular area of the software, this is an indication of potential future problems. It is the nature of defects to cluster and clump together. If it was defect ridden earlier, it will most likely continue to be defect prone.

One good approach to define where the risks are is to have several brainstorming sessions.

- Start with ideas, such as, what worries me about this project/application.

6 FEATURES TO BE TESTED

This is a listing of what is to be tested from the **USERS** viewpoint of what the system does. This is not a technical description of the software, but a USERS view of the functions.

Set the level of risk for each feature. Use a simple rating scale such as (H, M, L): High, Medium and Low. These types of levels are understandable to a User. You should be prepared to discuss why a particular level was chosen.

It should be noted that Section 4 and Section 6 are very similar. The only true difference is the point of view. Section 4 is a technical type description including version numbers and other technical information and Section 6 is from the User's viewpoint. Users do not understand technical software terminology; they understand functions and processes as they relate to their jobs.

7 FEATURES NOT TO BE TESTED

This is a listing of what is NOT to be tested from both the Users viewpoint of what the system does and a configuration management/version control view. This is not a technical description of the software, but a USERS view of the functions.

Identify WHY the feature is not to be tested, there can be any number of reasons.

- Not to be included in this release of the Software.
- Low risk, has been used before and is considered stable.
- Will be released but not tested or documented as a functional part of the release of this version of the software.

Sections 6 and 7 are directly related to Sections 5 and 17. What will and will not be tested are directly affected by the **levels of acceptable risk** within the project, and what does not get tested affects the **level of risk** of the project.

8 APPROACH (STRATEGY)

This is your overall test strategy for this test plan; it should be appropriate to the level of the plan (master, acceptance, etc.) and should be in agreement with all higher and lower levels of plans. Overall rules and processes should be identified.

- Are any special tools to be used and what are they?
- Will the tool require special training?
- What metrics will be collected?
- Which level is each metric to be collected at?
- How is Configuration Management to be handled?
- How many different configurations will be tested?
- Hardware
- Software
- Combinations of HW, SW and other vendor packages
- What levels of regression testing will be done and how much at each test level?
- Will regression testing be based on severity of defects detected?
- How will elements in the requirements and design that do not make sense or are untestable be processed?

If this is a master test plan the overall project testing approach and coverage requirements must also be identified.

Specify if there are special requirements for the testing.

- Only the full component will be tested.
- A specified segment of grouping of features/components must be tested together.

Other information that may be useful in setting the approach are:

- MTBF, Mean Time Between Failures - if this is a valid measurement for the test involved and if the data is available.
- SRE, Software Reliability Engineering - if this methodology is in use and if the information is available.

How will meetings and other organizational processes be handled?

9 ITEM PASS/FAIL CRITERIA

What are the Completion criteria for this plan? This is a critical aspect of any test plan and should be appropriate to the level of the plan.

- At the Unit test level this could be items such as:
  - All test cases completed.
  - A specified percentage of cases completed with a percentage containing some number of minor defects.
  - Code coverage tool indicates all code covered.
- At the Master test plan level this could be items such as:
  - All lower level plans completed.
  - A specified number of plans completed without errors and a percentage with minor defects.

This could be an individual test case level criterion or a unit level plan or it can be general functional requirements for higher level plans.

What is the number and severity of defects located?

- Is it possible to compare this to the total number of defects? This may be impossible, as some defects are never detected.
  - A defect is something that **may** cause a failure, and may be acceptable to leave in the application.
  - A failure is the result of a defect as seen by the User, the system crashes, etc.

10 SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

Know when to pause in a series of tests.

- If the number or type of defects reaches a point where the follow on testing has no value, it makes no sense to continue the test; you are just wasting resources.

Specify what constitutes stoppage for a test or series of tests and what is the acceptable level of defects that will allow the testing to proceed past the defects.

Testing after a truly fatal error will generate conditions that may be identified as defects but are in fact ghost errors caused by the earlier defects that were ignored.

11 TEST DELIVERABLES

What is to be delivered as part of this plan?

- Test plan document.
- Test cases.
- Test design specifications.
- Tools and their outputs.
- Simulators.
- Static and dynamic generators.
- Error logs and execution logs.
- Problem reports and corrective actions.

One thing that is not a test deliverable is the software itself that is listed under test items and is delivered by development.

12 REMAINING TEST TASKS

If this is a multi-phase process or if the application is to be released in increments there may be parts of the application that this plan does not address. These areas need to be identified to avoid any confusion should defects be reported back on those future functions. This will also allow the users and testers to avoid incomplete functions and prevent waste of resources chasing non-defects.

If the project is being developed as a multi-party process, this plan may only cover a portion of the total functions/features. This status needs to be identified so that those other areas have plans developed for them and to avoid wasting resources tracking defects that do not relate to this plan.

When a third party is developing the software, this section may contain descriptions of those test tasks belonging to both the internal groups and the external groups.

13 ENVIRONMENTAL NEEDS

Are there any special requirements for this test plan, such as:

- Special hardware such as simulators, static generators etc.
- How will test data be provided. Are there special collection requirements or specific ranges of data that must be provided?
- How much testing will be done on each component of a multi-part feature?
- Special power requirements.
- Specific versions of other supporting software.
- Restricted use of the system during testing.

14 STAFFING AND TRAINING NEEDS

Training on the application/system.

Training for any test tools to be used.

Section 4 and Section 15 also affect this section. What is to be tested and who is responsible for the testing and training.

15 RESPONSIBILITIES

Who is in charge?

This issue includes all areas of the plan. Here are some examples:

- Setting risks.
- Selecting features to be tested and not tested.
- Setting overall strategy for this level of plan.
- Ensuring all required elements are in place for testing.
- Providing for resolution of scheduling conflicts, especially, if testing is done on the production system.
- Who provides the required training?
- Who makes the critical go/no go decisions for items not covered in the test plans?

16 SCHEDULE

Should be based on realistic and validated estimates. If the estimates for the development of the application are inaccurate, the entire project plan will slip and the testing is part of the overall project plan.

- As we all know, the first area of a project plan to get cut when it comes to crunch time at the end of a project is the testing. It usually comes down to the decision, 'Let's put something out even if it does not really work all that well'. And, as we all know, this is usually the worst possible decision.

How slippage in the schedule will to be handled should also be addressed here.

- If the users know in advance that a slippage in the development will cause a slippage in the test and the overall delivery of the system, they just may be a little more tolerant, if they know it's in their interest to get a better tested application.
- By spelling out the effects here you have a chance to discuss them in advance of their actual occurrence. You may even get the users to agree to a few defects in advance, if the schedule slips.

At this point, all relevant milestones should be identified with their relationship to the development process identified. This will also help in identifying and tracking potential slippage in the schedule caused by the test process.

It is always best to tie all test dates directly to their related development activity dates. This prevents the test team from being perceived as the cause of a delay. For example, if system testing is to begin after delivery of the final build, then system testing begins the day after delivery. If the delivery is late, system testing starts from the day of delivery, not on a specific date. This is called dependent or relative dating.

17 PLANNING RISKS AND CONTINGENCIES

What are the overall risks to the project with an emphasis on the testing process?

- Lack of personnel resources when testing is to begin.
- Lack of availability of required hardware, software, data or tools.
- Late delivery of the software, hardware or tools.
- Delays in training on the application and/or tools.
- Changes to the original requirements or designs.

Specify what will be done for various events, for example:

- Requirements definition will be complete by January 1, 19XX, and, if the requirements change after that date, the following actions will be taken.
  - The test schedule and development schedule will move out an appropriate number of days. This rarely occurs, as most projects tend to have fixed delivery dates.
  - The number of test performed will be reduced.
  - The number of acceptable defects will be increased.
    - These two items could lower the overall quality of the delivered product.
  - Resources will be added to the test team.
  - The test team will work overtime.
    - This could affect team morale.
  - The scope of the plan may be changed.
  - There may be some optimization of resources. This should be avoided, if possible, for obvious reasons.

- You could just QUIT. A rather extreme option to say the least.

Management is usually reluctant to accept scenarios such as the one above even though they have seen it happen in the past.

The important thing to remember is that, if you do nothing at all, the usual result is that testing is cut back or omitted completely, neither of which should be an acceptable option.

18 APPROVALS

Who can approve the process as complete and allow the project to proceed to the next level (depending on the level of the plan)?

At the master test plan level, this may be all involved parties.

When determining the approval process, keep in mind who the audience is.

- The audience for a unit test level plan is different than that of an integration, system or master level plan.
- The levels and type of knowledge at the various levels will be different as well.
- Programmers are very technical but may not have a clear understanding of the overall business process driving the project.
- Users may have varying levels of business acumen and very little technical skills.
- Always be wary of users who claim high levels of technical skills and programmers that claim to fully understand the business process. These types of individuals can cause more harm than good if they do not have the skills they believe they possess.

19 GLOSSARY

Used to define terms and acronyms used in the document, and testing in general, to eliminate confusion and promote consistent communications.

SAMPLE MASTER TEST PLAN

1 TEST PLAN IDENTIFIER RS-MTP01.3

2 REFERENCES

None Identified.

3 INTRODUCTION

This is the Master Test Plan for the Reassigned Sales Re-write project. This plan will address only those items and elements that are related to the Reassigned Sales process, both directly and indirectly affected elements will be addressed. The primary focus of this plan is to ensure that the new Reassigned Sales application provides the same level of information and detail as the current system while allowing for improvements and increases in data acquisition and level of details available (granularity).

The project will have three levels of testing, Unit, System/Integration and Acceptance. The details for each level are addressed in the approach section and will be further defined in the level specific plans.

The estimated time line for this project is very aggressive (six (6) months), as such, any delays in the development process or in the installation and verification of the third party software could have significant effects on the test plan. The acceptance testing is expected to take one (1) month from the date of application delivery from system test and is to be done in parallel with the current application process.

4 TEST ITEMS

The following is a list, by version and release, of the items to be tested:

A. EXTOL EDI package, Version **3.0**

If a new release is available prior to roll-out it will not be used until after installation. It will be a separate upgrade/update project.

B. DNS PC EDI transaction package, Version **2.2**

If a new release is available prior to roll-out it will not be used until after installation. It will be a separate upgrade/update project.

C. Custom PC EDI transaction package (two distributors only).

D. New reassigned sales software, initial version to be Version **1.0**

A detailed listing of programs, databases, screens and reports will be provided in the system and detailed design documents.

E. Order Entry EDI interface software, Current version at time of pilot. Currently, version **4.1.**

F. Reassigned Sales System requirements document, SST_RQMT.WPD version **4.1**

G. Reassigned Sales System Design Document, SST_SYSD.WPD version **3.02**

H. Reassigned Sales Detail Design Document, SST_DTLD.WPD version **3.04**

5 SOFTWARE RISK ISSUES

There are several parts of the project that are not within the control of the Reassigned Sales application but have direct impacts on the process and must be checked as well.

A. The local AS/400 based vendor supplied EDI software package. This package will be providing all the reformatting support from the ANSI X12 EDI formats to the internal AS/400 data base file formats.

B. The PC based software package installed at each distributor's location (both custom written and vendor supplied) will be providing the formatting of the distributors data into the correct EDI X12 formats.

C. Backup and Recovery of the EDI transmission files, local databases and restart of the translation process, must be carefully checked.

D. The ability to restart the application in the middle of the process is a critical factor to application reliability. This is especially true in the case of the transmission files as once the data is pulled from the mail box it is no longer available there and must be protected locally.

E. Database security and access must be defined and verified, especially for files shared between the Order Entry application and the Reassigned Sales process. All basic security will be provided through the AS/400 systems native security process.

6 FEATURES TO BE TESTED

The following is a list of the areas to be focused on during testing of the application.

A. New EDI data acquisition process.

B. Redesigned On-line screens.

C. Redesigned/Converted reports.

D. New Automated Archive process.

E. Interface to the Order Entry system and data bases.

F. Computation of Sales Activity by region for commissions

7 FEATURES NOT TO BE TESTED

The following is a list of the areas that will not be specifically addressed. All testing in these areas will be indirect as a result of other testing efforts.

A. Non-EDI Order Entry processes.

Only the EDI interface of the Order Entry application will be verified. Changes to the EDI interface to support Reassigned Sales are not anticipated to have an impact on the Order Processing application. Order Entry is a separate application sharing the data interface only, orders will continue to process in the same manner.

B. Network Security and dial-in access.

Changes to include EDI transactions for reassigned sales will have no impact on the security aspects of the network or the EXTOL/EDI interface.

C. Operational aspects of the EDI process.

Changes to include EDI transactions for reassigned sales will have no impact on the operational aspects of the EXTOL/EDI interface.

D. PC based spreadsheet analysis applications using Reassigned Sales data.

These applications are completely under the control of the customer and are outside the scope of this project. The necessary data base format information will be provided to the customers to allow them to extract data. Testing of their applications is the responsibility of the application maintainer/developer.

E. Business Analysis functions using Reassigned Sales data.

These applications are completely under the control of the management support team and are outside the scope of this project. The necessary data base format information will be provided to the support team to allow them to extract data. Testing of their applications is the responsibility of the application maintainer/developer.

F. Marketing/Forecasting processes using Reassigned Sales data.

These applications are completely under the control of marketing and are outside the scope of this project. The necessary data base format information will be provided to marketing to allow them to extract data. Testing of their applications is the responsibility of the application maintainer/developer.

8 APPROACH

8.1 Testing Levels

The testing for the Reassigned Sales project will consist of Unit, System/Integration (combined) and Acceptance test levels. It is hoped that there will be at least one full time independent test person for system/integration testing. However, with the budget constraints and time line established; most testing will be done by the test manager with the development teams participation.

UNIT Testing will be done by the developer and will be approved by the development team leader. Proof of unit testing (test case list, sample output, data printouts, defect information) must be provided by the programmer to the team leader before unit testing will be accepted and passed on to the test person. All unit test information will also be provided to the test person.

SYSTEM/INTEGRATION Testing will be performed by the test manager and development team leader with assistance from the individual developers as required. No specific test tools are available for this project. Programs will enter into System/Integration test after all critical defects have been corrected. A program may have up to two Major defects as long as they do not impede testing of the program (I.E. there is a work around for the error).

ACCEPTANCE Testing will be performed by the actual end users with the assistance of the test manager and development team leader. The acceptance test will be done in parallel with the existing manual ZIP/FAX process for a period of one month after completion of the System/Integration test process.

Programs will enter into Acceptance test after all critical and major defects have been corrected. A program may have one major defect as long as it does not impede testing of the program (I.E. there is a work around for the error). Prior to final completion of acceptance testing all open critical and major defects MUST be corrected and verified by the Customer test representative.

A limited number of distributors will participate in the initial acceptance test process. Once acceptance test is complete, distributors will be added as their ability to generate the required EDI data is verified and checked against their FAX/ZIP data. As such, some distributors will be in actual production and some in parallel testing at the same time. This will require careful coordination of the control tables for the production system to avoid posting test data into the system.

8.2 Configuration Management/Change Control

Movement of programs from the development portion of the 'RED' system to the test portion of the 'RED' system will be controlled through the existing Configuration Management application process, 'EXTRACT'. This will ensure that programs under development and those in full test will have the same version controls and tracking of changes. The same extract process will be used to migrate the programs from the Development/Test 'RED' system to the production 'BLUE' system once all testing has been completed according to published plans and guidelines.

All Unit and initial system testing will be performed on the Development AS/400 'RED' system. Once the system has reached a reasonable level of stability, no critical or major defects outstanding, initial pilot testing will be done on the production AS/400 'BLUE' system. All testing done on the 'BLUE' system will be done in a parallel mode will all controls set to prevent actual updating of the production files.

This will allow some early testing of the numbers received through the old ZIP/FAX process and the higher level of detail received through the new EDI process. This will also help identify potential problems with the comparison of the two sets of numbers.

All changes, enhancements and other modification requests to the system will be handled through the published change control procedures. Any modifications to the standard procedures are identified in the project plan change control section.

8.3 Test Tools

The only test tools to be used are the standard AS/400 provided utilities and commands.

A. The Program Development Manager (PDM) will be used as the source version configuration management tool in conjunction with the in-house check-in/check-out control utility. The check-in/out utility is part of each developers standard AS/400 access menu.

B. The initial prototypes for the new screens will be developed using the AS/400 Screen Design Aid (SDA). The initial layout and general content of the screens will be shown to the sales administration staff prior to proceeding with testing and development of the screens.

C. All editing, compiling and debugging will be done using the Source Entry Utility (SEU).

D. Data acquisition will be from actual production files where available using the AS/400 data base copy command CPYF and it's various functions. Additional data will be created and modified as needed using the Data File Utility (DFU). No changes will ever be made to actual production files under any circumstances.

E. Initial data for EDI testing will be done using one or two beta sites and replicating the data at the mailbox location or locally in the control files, to create high volume data and to simulate multiple distributors sending in data.

8.4 Meetings

The test team will meet once every two weeks to evaluate progress to date and to identify error trends and problems as early as possible. The test team leader will meet with development and the project manager once every two weeks as well. These two meetings will be scheduled on different weeks. Additional meetings can be called as required for emergency situations.

8.5 Measures and Metrics

The following information will be collected by the Development team during the Unit testing process. This information will be provided to the test team at program turnover as well as be provided to the project team on a biweekly basis.

1. Defects by module and severity.
2. Defect Origin (Requirement, Design, Code)
3. Time spent on defect resolution by defect, for Critical & Major only. All Minor defects can be totaled together.

The following information will be collected by the test team during all testing phases. This information will be provided on a biweekly basis to the test manager and to the project team.

1. Defects by module and severity.
2. Defect Origin (Requirement, Design, Code)
3. Time spent on defect investigation by defect, for Critical & Major only. All Minor defects can be totaled together.
4. Number of times a program submitted to test team as ready for test.
5. Defects located at higher levels that should have been caught at lower levels of testing.

9 ITEM PASS/FAIL CRITERIA

The test process will be completed once the initial set of distributors have successfully sent in reassigned sales data for a period of one month and the new EDI data balances with the old ZIP/FAX data received in parallel. When the sales administration staff is satisfied that the data is correct the initial set of distributors will be set to active and all parallel stopped for those accounts.

At this point the next set of distributors will begin the parallel process, if not already doing so. Only the initial set of distributors must pass the data comparison test to complete the testing, at that point the application is considered live. All additional activations will be on an as ready basis. When a distributor is ready, and their data is verified, they will then also be activated.

10 SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

A. No Distributors are ready for testing at pilot initiation.

The pilot project will be delayed until at least three Distributors are ready to initiate the pilot process. No additional elements will be added to the Reassigned Sales project during this delay.

B. Unavailability of two EDI mail boxes.

In the event two production lines and mail box facilities cannot be obtained the current single production line and mail box will continue to be used until a second line becomes available. This will necessitate careful coordination between the Order Entry department and the Reassigned Sales group.

C. Distributor PC EDI software delays.

In the event of a delay in the delivery or availability of the PC software package, the only major delay will be in pilot testing. Unit, Integration and Systems testing can continue using limited data until such time as the PC software is ready.

This will also add time to the lower levels of testing as full complete testing cannot be done without reasonable amounts of data. The data can only be derived from actual transmissions from the PC software package.

11 TEST DELIVERABLES

Acceptance test plan

System/Integration test plan

Unit test plans/turnover documentation

Screen prototypes

Report mock-ups

Defect/Incident reports and summaries

Test logs and turnover reports

12 REMAINING TEST TASKS

TASK	Assigned To	Status
Create Acceptance Test Plan	TM, PM, Client	
Create System/Integration Test Plan	TM, PM, Dev.	
Define Unit Test rules and Procedures	TM, PM, Dev.	
Define Turnover procedures for each level	TM, Dev	
Verify prototypes of Screens	Dev, Client, TM	
Verify prototypes of Reports	Dev, Client, TM	

13 ENVIRONMENTAL NEEDS

The following elements are required to support the overall testing effort at all levels within the reassigned sales project:

A. Access to both the development and production AS/400 systems. For development, data acquisition and testing.

B. A communications line to the EDI mailbox facility. This will have to be a shared line with the Order Entry process as only one mailbox is in use. There will have to be a coordinated effort to determine how often to poll the mailbox as the order entry process requires that data be accessed every hour and the sales process really only needs to be pulled once a day.

C. An installed and functional copy of the AS/400 based EDI vendor package.

D. At least one distributor with an installed copy of the PC based EDI vendor package for sales data.

E. Access to the master control tables (data bases) for controlling the production/testing environment on both production and development systems.

F. Access to the nightly backup/recovery process.

14 STAFFING AND TRAINING NEEDS

It is preferred that there will be at least one (1) full time tester assigned to the project for the system/integration and acceptance testing phases of the project. This will require assignment of a person part time at the beginning of the project to participate in reviews etc... and approximately four months into the project they would be assigned full time. If a separate test person is not available the project manager/test manager will assume this role.

In order to provide complete and proper testing the following areas need to be addressed in terms of training.

A. The developers and tester(s) will need to be trained on the basic operations of the EDI interface. Prior to final acceptance of the project the operations staff will also require complete training on the EDI communications process.

B. The sales administration staff will require training on the new screens and reports.

C. At least one developer and operations staff member needs to be trained on the installation and control of the PC based distributors EDI package. The distributors personnel will also have to be trained on the PC based package and its operational characteristics.

15 RESPONSIBILITIES

	TM	PM	Dev Team	Test Team	Client
Acceptance test Documentation & Execution	X	X		X	X
System/Integration test Documentation & Exec.	X		X	X	
Unit test documentation & execution	X		X	X	
System Design Reviews	X	X	X	X	X
Detail Design Reviews	X	X	X	X	
Test procedures and rules	X	X	X	X	
Screen & Report prototype reviews			X	X	X
Change Control and regression testing	X	X	X	X	X

The development team leader will be responsible for the verification and acceptance of all unit test plans and documentation.

The project manager/test manager is responsible for all test plans and documentation.

The entire project team will participate in the review of the system and detail designs as well as review of any change requests that are generated by the user or as a result of defects discovered during development and testing. The sales administration staff is also required to participate in the initial high-level system review.

The sales administration staff will provide a person, as required, throughout the project to verify test results and answer questions as they arise. This person will also be responsible for participating in the execution of the acceptance test plan.

16 SCHEDULE

Time has been allocated within the project plan for the following testing activities. The specific dates and times for each activity are defined in the project plan time line. The persons required for each process are detailed in the project time line and plan as well. Coordination of the personnel required for each task, test team, development team, management and customer will be handled by the project manager in conjunction with the development and test team leaders.

A. Review of Requirements document by test team personnel (with other team members) and initial creation of Inventory classes, sub-classes and objectives.

B. Development of Master test plan by test manager and test with time allocated for at least two reviews of the plan.

C. Review of the System design document by test team personnel. This will provide the team with a clearer understanding of the application structure and will further define the Inventory classes, sub-classes and objectives.

D. Development of System/Integration and Acceptance test plans by test manager and other essential personnel with time allocated for at least two reviews of the plans.

E. Review of the Detail design document(s) by test team personnel as required. This will provide the team with a clearer understanding of the individual program structure and will further define the Inventory classes, sub-classes and objectives.

F. Unit test time within the development process.

G. Time allocated for both System/Integration and Acceptance test processes.

17 PLANNING RISKS AND CONTINGENCIES

A. Limited Reassigned Sales staff.

The Reassigned Sales administration staff currently has two positions unfilled. As a result of this staff shortage there may be delays in getting staff to review appropriate documents and to participate in the Acceptance test process. Should client staff become a problem, the appropriate dates for reviews and acceptance testing will slip accordingly. No attempt will be made to bypass any part of the review and testing processes.

However, if acceptable to the Reassigned Sales staff administrator, a member if the test team my be available to act as the client's representative for part of the Acceptance test itself. The reviews of the screens and reports must have Client participation and approval.

18 APPROVALS

Project Sponsor - Steve Sponsor	
Development Management - Ron Manager	
EDI Project Manager - Peggy Project	
RS Test Manager - Dale Tester	
RS Development Team Manager - Dale Tester	
Reassigned Sales - Cathy Sales	
Order Entry EDI Team Manager - Julie Order	

1 SELECTING AND IMPLEMENTING A TOOL

This chapter gives general guidance about selecting and evaluating commercial tools. It is intended to provide a starting point for tool assessment. Although it is not as detailed or specific as a tailored report prepared by a consultant, it should enable you to plan the basics of your own tool selection and evaluation process.

It is easy to make the mistake of considering only tool function, and not the other success factors related to the organisation where the tool will be used; this is one reason that expensive tools end up being unused only a few months after purchase. Following the advice given in this chapter should help you to avoid some of these problems.

There are a surprising number of steps in the tool selection process. The following diagram is an overview of the whole process.

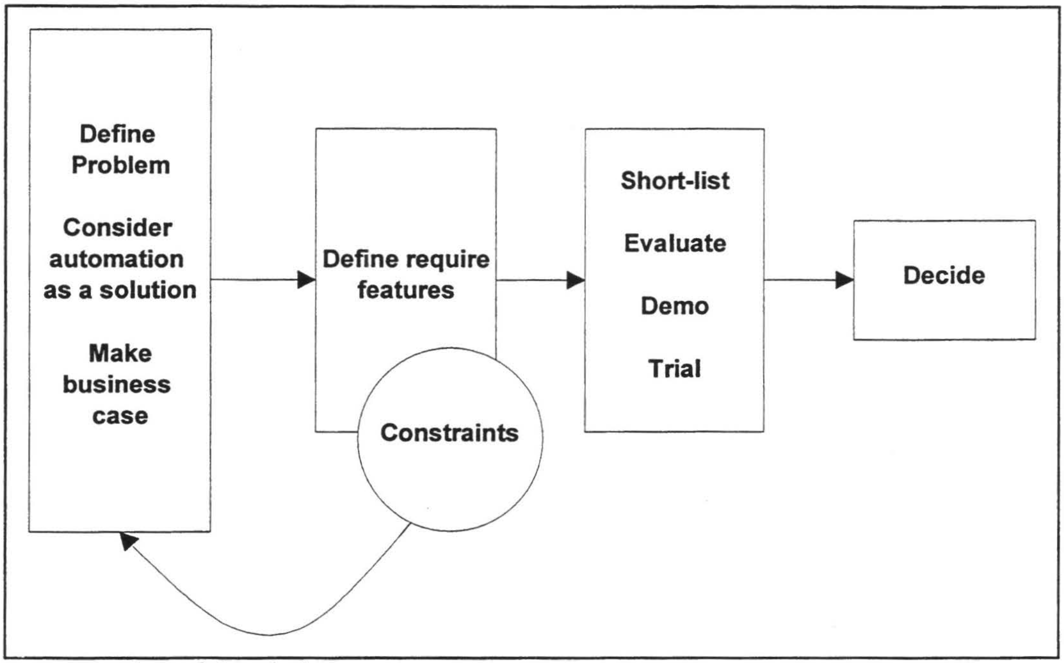


Figure 4.1 Overview of the selection process

1.1 Where to start

You are probably reading this report because you want to make your testing process more efficient through the use of a software testing tool. However, there is a wide variety of tools available; which one(s) should you buy?

There are a number of different types of testing tools on the market, and they serve different purposes. Buying a capture/replay tool will not help you measure test coverage; a static analysis tool will not help in repeating regression tests.

You also need to consider the environment where the testing tool will be used: a mainframe tool is no use if you only have PCs.

The skills of the people using the tool also need to be taken into account; if a test execution tool requires programming skills to write test scripts, it would not be appropriate for use by end-users only.

These considerations are critical to the successful use of the tool; if the wrong tool is selected, whether it is the wrong tool for the job, the environment or the users, the benefits will not be achieved.

1.2 The tool selection and evaluation team

Someone should be given the responsibility for managing the selection and evaluation process. Generally a single individual would be authorised to investigate what tools are available and prepare a shortlist, although there could be several people involved. The researchers need to have a reasonable idea of what type of tool is needed, who within the organisation would be interested in using it, and what the most important factors are for a tool to qualify for the shortlist.

After the shortlist is prepared, however, it is wise to involve a number of people in the evaluation process. The evaluation team should include a representative from each group planning to use the tool, and someone of each type of job function who would be using it. For example, if non-technical end-users will be using a test execution tool to run user acceptance tests, then a tool that needs programming skills would be excluded and an end-user should be on the evaluation team. The usability of a tool has a significant effect on whether the tool becomes accepted as part of the testing process.

If the evaluation team becomes involved in a trial scheme, the intended user must make the recommendation as to the tool's usability. However, the team may need to have access to a systems support resource. The role of the support person is to assist in overcoming technical problems which appear important to the user or developer but are actually quite easily overcome by a technician.

The selection and evaluation team may also go on to become the implementation team, but not necessarily.

1.3 What are the problems to be solved?

The starting point for tool selection is identifying what problem needs to be solved, where a tool might provide a solution. Some examples are:

- tests which are currently run manually are labour-intensive, boring and lengthy
- tests which are run manually are subject to inconsistencies, owing to human error in inputting tests
- we need knowledge of the completeness or thoroughness of our tests; the measurement of the amount of software tested with a test suite is difficult or impossible to compute manually
- paper records of tests are cumbersome to maintain, leading to tests being repeated or omitted
- when small changes are made to the software, extensive regression tests must be repeated and there is not time to do it manually
- setting up test data or test cases is repetitive and 'mechanical': the testers find it uninteresting and make too many 'simple' errors
- comparison of test results is tedious and error-prone
- errors are found during testing which could have been detected before any tests were run by examining the code carefully enough
- users find too many errors that could have been found by testing.

The current testing process must be sufficiently well-defined that it is easy to see the areas where automated improvement would actually help.

1.4 Is a tool the right solution?

Software testing tools are one way to approach these types of problems, but are not the only way. For example, code inspection could be used to address the problem of detecting errors before test execution. Better organisation of test documentation and better test management procedures would address the problem of omitting or repeating tests. Considering whether all the repetitive 'mechanical' test cases are really necessary may be more important for test effectiveness and efficiency than blindly automating them. The use of a testing tool will not help in finding more errors in testing unless the test design process is improved, which is done by training, not by automation.

An automated solution often 'looks better' and may be easier to authorise expenditure for than addressing the more fundamental problems of the testing process itself. It is important to realise that the tool will not correct a poor process without additional attention being paid to it. It is possible to improve testing practices alongside implementing the tool, but it does require conscious effort.

However, we will assume that you have decided, upon rational consideration of your own current situation (possibly with some tool readiness assessment advice from an outside organisation), that a testing tool is the solution you will be going for.

1.5 How much help should the tool be?

Once you have identified the area of testing you want the tool to help you with, how will you be able to tell whether any tool you buy has actually helped? You could just buy one and see if everyone feels good about it, but this is not the most rational approach. A better way is to define measurable criteria for success for the tool. For example, if the length of time taken to run tests manually is the problem, how much quicker should the tests be run using a tool?

Setting measurable criteria is not difficult to do, at least to obtain a broad general idea of costs. A general idea is all that is necessary to know whether the tool will be cost-justified. A realistic measurable criterion for a test execution tool might be set out as follows:

> *Manual execution of tests currently takes 4 man-weeks. In the first 3 months of using the tool, 50–60 per cent of these tests should be automated, with the whole test suite run in 2–2½ man-weeks. Next year at this time we aim to have 80 per cent of the tests automated, with the equivalent test suite being run in 4 man-days.*

An approach to measuring the potential savings of a test coverage tool might be:

> *We currently believe that our test cases 'completely test' our programs, but have no way of measuring coverage. As an experiment on a previously tested (but unreleased) program, rerun the dynamic tests using a coverage measurement tool. We will almost certainly find that our tests reached less than 100 per cent coverage. Based on the tool's report of the unexecuted code we can devise and run additional test cases. If these additional test cases discover errors – serious ones that are deemed likely to have appeared sometime in the future in live running – then the tool would make a saving by detecting those errors during testing, which is less costly than in live running. The potential savings are the difference between the cost of an error found in live running: say £5000 for a modest error that must be corrected, and an error found during testing, say £500.*

A similar approach could be applied to determining whether a static analysis tool is worth using:

> *Run the static analysis tool on a group of programs that have been through dynamic testing but are not yet released. Evaluate (in the opinion of a senior analyst) the cost of the genuine errors detected. For those errors which were also found in the original dynamic testing, the static analyser might not save the cost of running the dynamic test (because you do not design your dynamic tests assuming that errors have been found previously), but it might save the cost of all the diagnosis and rerunning that dynamic testing entails. More interesting is the cost of those errors which were only detected by static test but which would have caused a problem in live running. The cost of detecting the error through static analysis is likely to be one-hundredth the cost of finding it in live running.*

When looking at the measurable benefits it is best to be fairly conservative about what could be accomplished. When a tool is used for the first time it always takes much longer than when people are experienced in using it, so the learning curve must be taken into account. It is important to set realistic goals, and not to expect miracles. It is also important that the tool is used correctly, otherwise the benefits may not be obtained.

If you find that people are prepared to argue about the specific numbers which you have put down, ask them to supply you with more accurate figures which will give a better-quality evaluation. Do not spend a great deal of time 'polishing' your estimates: the tool evaluation process should be only as long as is needed to come to a decision, and no longer. Your estimates should reflect this granularity.

1.6 How much is this help worth?

The measurable criteria that you have identified as achievable will have a value to your organisation; it is important to quantify this value in order to compare the cost of the tool with the cost saved by the benefits. One of the simplest ways to quantify the benefits is to measure the saving of time and multiply that by approximate staff costs.

For example, if regression tests which normally take 4 man-weeks manually can be done in 2 man-weeks we will save 2 man-weeks of effort whenever those tests are run. If they are run once a quarter, we will save 8 man-weeks a year. If they are run once a month, we will save 24 man-weeks a year. (If they are only run once a year we will only save 2 man-weeks in that year.) If a man-week is costed at say, £2,000, we will save respectively £16,000, £48,000 or £4,000.

The savings that can be achieved can be taken advantage of by putting the people into more productive work, on development, enhancements or better test design.

There will also be other benefits, which may be very difficult if not impossible to quantify but which should also be mentioned. The risk of an embarrassing public release may be reduced, for example, but it may not be possible to put a monetary value on this. Morale is likely to improve, which is likely to result in an increase in productivity, but it may not be possible or desirable to separate this from the productivity increase from using the tool. There may be some things that are not even possible to do manually, which will not be discovered until the tool has been in use; these unanticipated benefits cannot be quantified because no one realised them at the time.

Of course this is a very simplistic start to building a proper business case for the tool, but it is essential that some first attempt is made to quantify the benefits, otherwise you will not be able to learn from your tool evaluation experience for next time.

2 TOOL REQUIREMENTS

2.1 What tool features are needed to meet requirements?

The next step is to begin to familiarise yourself with the general capabilities of tools of the type you want.

Which of the features listed are the most important ones to meet the needs and objectives for the tool in your current situation? For example, if you want to improve the accuracy of test results comparison, a capture/replay tool without a comparator would not help.

Make a list of the features, classified as 'essential', 'desirable' and 'don't matter'. The essential features list would rule out any tool which did not provide all of the things on that list, and the desirable features would be used to discriminate among those tools that provide all the essential features.

Note that your feature list will change as you progress in evaluating tools. You will almost certainly discover new features that you think are desirable as you go through the evaluation process. The tool vendors are sure to point out features they can supply but which you did not specifically request. Other tool users may recommend a feature as essential because of their experience, which you may not have thought was so important. For example, you may not consider the importance of being able to update your test scripts whenever the software changes because you are concentrating on the use of the execution tool for capturing tests the first time. However, this may be a significant 'running cost' for using the testing tool in the future. It is also possible that a feature that you thought was desirable is not required owing to the way other features are implemented.

As well as the functions that the tool performs, it is important to include some grading of usability as a feature for evaluation. Tools that have sound technical features, but are difficult to use, frequently become shelfware.

2.2 What are the constraints?

2.2.1 Environmental constraints

Testing tools are software packages and therefore may be specific to particular hardware, software or operating systems. You would not want a tool that runs only on a VAX VMS system if you have an IBM MVS system and no possibility of acquiring or using anything else.

Most people look for a tool which will run on the environment in which they are developing or maintaining software, but that is not the only possibility. A number of tools can run on a PC, for example, and can execute tests running on a different computer. Even debug and coverage measurement tools can work in a 'host–target' or client–server environment.

Having to acquire additional hardware is sometimes more of a psychological barrier than a technical or economic one. In your tool selection process, especially if there are not many for your 'home' environment, it is worth considering tools based on a separate environment.

However, you may need to acquire extra hardware even for a tool that runs on your own current environment, for example extra disk space to store test scripts.

Make sure that you find out exactly what the tool requires in terms of hardware and software versions. For example, you would not want to discover at installation time that you needed to have an operating system upgrade or additional memory before the tool can work. Have you considered security aspects? Do you need a separate language compiler for the test scripts?

2.2.2 Commercial supplier constraints

The company that you buy the tool from will be an important factor for your future testing practices. If you have problems with the tool, you will want them sorted out quickly and competently. If you want to get the best from the tool, you will want to take advantage of their expertise. You may want to influence the future development of the tool to provide for those needs which are not currently met by it.

There are a number of factors that you should take into consideration in evaluating the tool vendor's organisation:

- Is the supplier a *bona fide* company?
- How mature are the company and the product? If the company is well established this gives confidence, but if the product has not changed significantly in recent years it may be getting rather out of date. Some organisations will feel that they need to buy products from the product vendor who sets the trend in the marketplace. Some organisations will be wary of new product companies, but there may be instances when a brand new organisation or product may be an unknown quantity but may be just what you need at just the right time. A new vendor may be much more eager to please their first customers;
- Is there adequate technical support? What would their response be to major or minor problems? Does the vendor run a help desk? What hours is help available? (If your vendor is in California and you are in Europe, there will be no overlap of their working day with yours!) What training courses are provided? How responsive are they to requests for information?
- How many other people have purchased or use this tool? You may or may not want to be the very first commercial user of a new tool. Can you talk to any other users? Is there a user group, and when does it meet and who controls it? Will they provide a reference site for you to talk to?
- What is the tool's history? Was it developed to support good internal testing practices, to meet a specific client need, or as a speculative product? How many releases have there been to date, and how often is the tool updated? How many open faults are there currently reported?

Your relationship with the tool vendor starts during the selection and evaluation phase. If there are problems with the vendor now (when they want your money), there are likely to be even more problems later.

2.2.3 Cost constraints

Cost is often the most stringent and most visible constraint on tool selection. The purchase price may only be a small factor in the total cost to the organisation in fully implementing the tool. Cost factors include:

- purchase or lease price
- cost basis (per seat, per computer etc.)
- cost of training in the use of the tool
- any additional hardware needed (e.g. a PC, additional disk space or memory)
- support costs
- any additional costs, e.g. consultancy to ensure the tool is used in the best way.

2.2.4 Other constraints

Tool quality factors may include:

- How many people can use the tool at the same time? Can test scripts be shared?

- What skill level is needed to use the tool? How long does it take to become proficient? Are programming skills needed to write test scripts?
- What documentation is supplied? How thorough is it? How usable is it? Are there 'quick reference guides', for example?

There may well be other constraints which override all of the others, for example 'political' factors, such as having to buy the same tool that the parent company uses (e.g. an American parent enforces an American tool as its standard), or a restriction against buying anything other than a locally supported tool, perhaps limiting the choice to a European or British, French or German tool. It is frustrating to tool selectors to discover these factors late on in the selection process.

2.3 Constructing the shortlist

Use the cross-references in this report to find the tools that meet your environmental requirements and provide the features that are essential for you. Read the descriptions in the tools pages. This should give you enough information to know which tools listed in this report can go on your shortlist for further evaluation.

If there are more than six or seven tools that are suitable for you, you may want to do some initial filtering using your list of desirable features so that you will be looking at only three or four tools in your selection process.

If no or not enough suitable tools are found in this report, the search could be widened to other countries (e.g. the USA).

Other sources of information include pre-commercial tools (if you can find out about them). It is worth asking your current hardware or system software supplier if they know of any tools that meet your needs. If you are already using a CASE tool, it would be worth asking your vendor about support for testing, either through future development of their tool or by linking to an existing CAST tool. Conferences and exhibitions are where new vendors often go to announce a new tool. In particular the EuroSTAR conference is the prime showcase for testing tools.

The possibility of in-house development of a special-purpose tool should also be assessed. Do not forget to consider any existing in-house written tools within your own organisation that may be suitable for further development to meet your needs. The true cost of in-house development, including the level of testing and support needed to provide a tool of adequate quality, will be significant. It is generally much more than the cost of a commercial tool, but an in-house written tool will be more directly suitable to your own needs. For example, it can help to compensate for a lack of testability in the software under test. A purchased tool may need additional tailoring in order to meet real needs, and this can be expensive.

Another possibility is to use a 'meta-tool' to develop a new tailored tool in a short time. A meta-tool provides software for building software tools quickly, using the existing foundation of a standardised but highly tailorable user interface, graphical editor and text editor. It could enable a new testing tool, tailored to a specific organisation, to be built within a few months.

Summary of where to look for tools

- Existing environment (this report)
- PC-based (this report)
- In-house prototype for development
- Future environment of likely vendor
- Hardware/software vendor
- CASE tool vendor
- Meta-tool vendor
- World market
- Conferences and exhibitions

3 TOOL EVALUATION

3.1 Evaluating the shortlisted candidate tools

3.1.1 Research and compare tool features

Contact the vendors of the shortlisted tools and arrange to have information sent (if you have not done this already). Study the information and compare features. Request further information from the vendors if the literature sent does not explain the tool function clearly enough.

This is the time to consult one or more of the publications which have evaluated testing tools, if the ones you are interested in are covered in such a report. The cost of such reports should be compared to the cost of someone's time in performing similar evaluations, and the cost of choosing the wrong tool because you did not know about something which was covered in published material. (Do not forget to allow time to read the report.)

Ask the shortlisted vendors to give you the names of some of their existing customers as reference. Contact the reference sites from each shortlisted vendor and ask them a number of questions about the tool. For example, why they bought this tool, how extensively it is now used, whether they are happy with it, what problems they have had, their impression of the vendor's after-sales support service, how the tool affected their work, what benefits the tool gave them, and what they would do differently next time they were buying a tool. Remember that reference sites are usually the vendor's best customers, and so will be likely to be very happy with the tool. Their environment is different from yours, so the benefits or problems which they have had may well not be the same as the ones which are important to you. However, the experience of someone else who bought a tool for similar reasons to yours is invaluable and well worth pursuing.

Many vendors are aware that a tool does not always add up to a total solution and are keen to present it as part of a more comprehensive offering, often including consultancy and training beyond just their product. They usually understand the issues covered in this chapter and the next, because bad selection and bad implementation of their tools gives them a bad reputation. Because the vendors have good experience in getting the best out of their tools, their solutions may enhance the tools significantly and are worth serious examination. Nevertheless, it is always worth bearing in mind that the tool supplier is ultimately trying to persuade you to buy his product.

At any point in the selection and tool evaluation process it may become clear which tool will be the best choice. When this happens, any further activities will not influence the choice of tool but may still be useful in assessing in more detail how well the chosen tool will work in practice. It will either detect a catastrophic mismatch between the selected tool and your own environment, or will give you more confidence that you have selected a workable tool.

3.1.2 Tool demonstrations: preparation

Before contacting the vendor to arrange for a tool demonstration, some preparatory work will help to make your assessment of the competing tools more efficient and unbiased. Prepare two test case suites for tool demonstration:

- one of a normal 'mainstream' test case
- one of a worst-case 'nightmare' scenario.

Rehearse both tests manually, in order to discover any bugs in the test scenarios themselves. Prepare evaluation forms or checklists:

- general vendor relationship (responsiveness, flexibility, technical knowledge)

- tool performance on your test cases. Set measurable objectives, such as time to run a test on your own (first solo flight), time to run a reasonable set of tests, time to find an answer to a question in the documentation.

It is important that the tools be set up and used on your premises, using your configurations, and we recommend this, if at all possible, for the demonstration. We have had clients report to us that they found this single step to be extremely valuable, when they discovered that their prime candidate tool simply would not run in their environment! Of course, the vendor may be able to put it right but this takes time, and it is better to know about it before you sign on the dotted line, not after.

Invite the vendors of all shortlisted tools to give demonstrations within a short time-frame, for example on Monday, Wednesday and Friday of the same week. This will make sure that your memory of a previous tool is still fresh when you see a different one.

Give vendors both of your test cases in advance, to be used in their demo. If they cannot cope with your two cases in their demo, there probably is not much hope of their tool being suitable. However, be prepared to be flexible about your prepared tests. The tool may be able to solve your underlying problem in a different way than you had pictured. If your test cases are too rigid, you may eliminate a tool which would actually be very suitable for you.

Find out what facilities the vendors require and make sure they are available. Prepare a list of questions (technical and commercial) to ask on the demo day, and prepare one more test case suite to give them on the day. Allow time to write up your reactions to each of the tools, say at the end of each day.

3.1.3 Tool demonstrations from each vendor

Provide facilities for the vendor's presentation and their demonstration. Listen to the presentation and ask the questions you had prepared.

Observe their running of your prepared test case suites. Try your own 'hands-on' demonstration of your prepared test case suites and the new one for the day. Have a slightly changed version of the software being tested, so that the test suite needs to be modified to test the other version. Have the vendors edit the scripts if they insist, but it is better to edit them yourself with their assistance, so that you can see how much work will be involved in maintaining scripts.

Ask (and note) any more questions which occur to you. Note any additional features or functions which you had not realised this tool provided. Note any features or functions which you thought it did provide but does not, or not in the form you had thought.

Try to keep all demonstrations the same as far as possible. It is easy for the last one to incorporate improvements learned during the other demonstrations, but this is not fair to the first one. Save new ideas for use in the competitive trial.

Thank and dismiss the vendor. Write up your observations and reactions to this tool.

3.1.4 Post-demonstration analysis

Ask of the vendors you saw first any questions which occurred to you when watching a later vendor's presentation or demonstration. This will give the fairest comparison between the tools.

Assess tool performance against measurable criteria defined earlier, taking any special circumstances into account. Compare features and functions offered by competing tools. Compare non-functional attributes, such as usability. Compare the commercial attributes of vendor companies.

If a clear winner is now obvious, select the winning tool. Otherwise select two tools for final competitive trial. Write to the non-selected vendors giving the reason for their elimination.

3.1.5 Competitive trial

If it is not clear which tool is the most appropriate for you at this point, an in-house trial or evaluation will give a better idea of how you would use the tool for your systems.

Most tool vendors will allow short-term use of the tool under an evaluation licence, particularly for tools which are complex and represent a major investment. Such licences will be for a limited period of time, and the evaluating unit must plan and prepare for that evaluation accordingly.

It is all too easy to acquire the tool under an evaluation licence only to find that those who really ought to be evaluating it are tied up in some higher-priority activity during that time. If they are not able or willing to make the time available during the period of the licence to give the tool more than the most cursory attention, then the evaluation licence will be wasted.

The preparation for the trial period includes the drafting of a test plan and test suites to be used by all tools in the trial. Measurable success criteria for the evaluated tools should be planned in advance, for example length of time to record a test or to replay a test, and the number of discrepancies found in comparison (real, extraneous and any missed). Attending a training course for each tool will help to ensure that they will be used in the right way during the evaluation period.

When the competitive trial evaluation is performed, each tool will be installed and the selected tests will be run using each tool. Ensure that all mandatory requirements are met. Use the discriminatory factors and the ranked list of features and functions to give an objective score to each tool. Measure the success criteria and estimate the potential savings based on the business case for tool purchase which you had prepared earlier.

For a test execution tool such as a capture/replay tool, it may be a good idea to run the competitive trial on some software which is actively being developed. Although it is normally not recommended to capture scripts while the software is still unstable, doing so for the competitive trial will give a realistic insight into the test maintenance task.

Finally, analyse the results of the trial. Compare the features and functions offered by the tools. Compare the non-functional attributes, especially usability and general 'gut feel'. Compare the commercial attributes of vendor companies, and experience in tool installation and support during the trial. Compare other users' experiences in using the tool, from reference sites.

Assess any additional tool features which were not evaluated in this trial. A sophisticated tool may provide significant further functions or features which would allow you to 'grow into' it later on, rather than acquiring other tools to provide new features. A tool which is easy to learn in a week may look better at the end of the trial but may reach its limits within a fairly short time.

3.2 Making the decision

Having spent a considerable amount of effort in assessing the candidate, the end result would normally be a recommendation to purchase the tool the selectors felt would best meet the needs of the organisation.

Before making this recommendation, assess the business case: will the potential savings from this tool give a good return on investment, including purchase/lease price, training costs and ongoing internal tool costs? The likely benefits need to be clearly communicated, so that expectations for the benefits are realistic. Deciding not to go ahead with purchasing any of the tools investigated could be the best economic decision at this time; do not be afraid to make it, if a tool is not justified.

It can be very difficult to choose the ultimate best tool. If a clear-cut winner is still not obvious, then you are likely to be equally successful with either tool and commercial factors such as the status of the tool supplier, the flexibility of licensing arrangements or the company most willing to be flexible on some other aspect of the purchase will become the deciding factor. If the commercial factors have not swayed the decision to a particular vendor, then further agonising over the choice will not be profitable. If the technical and other issues cannot make the decision, then trust intuitive feelings about the people involved.

Inform the winning vendor, and let them know what your organisation's purchasing procedure is. For example, if a management meeting has to approve the expenditure and purchase orders have to be raised, it may be several weeks or even months before you can begin using the tool.

It is also an important courtesy to inform the loser(s), and to let them know why you have decided on another tool.

The selection and evaluation process should be limited in time, but once the winning tool has been identified, the more significant process of implementing the tool within the organisation can start being planned. This report has less detail about implementation than selection, but in practice implementation is much more significant.

4 IMPLEMENTATION

4.1 Starting implementation

Once a tool has been chosen, the real work starts. Although it is important to choose carefully to get the right tool, success in its use is by no means guaranteed. This chapter covers the critical factors in ensuring that the potential benefits from the purchased tool are actually achieved.

Tools tend towards 'shelfware' without a proper implementation process.

If the process of tool selection is one of gradually narrowing down the choices, the implementation process is the reverse: it is a process of gradually widening the tool's acceptance, use and benefits, as illustrated in the following diagram.

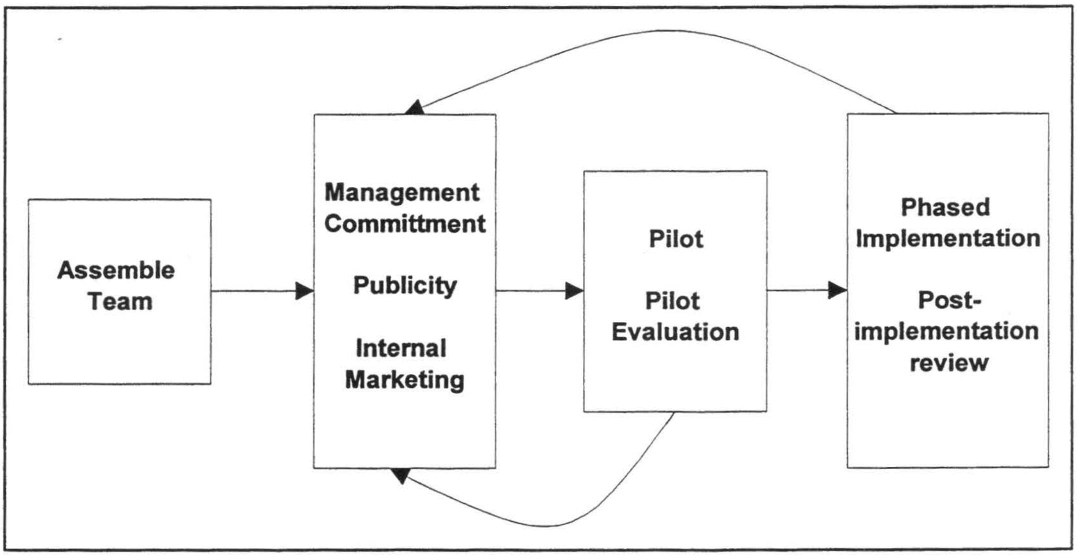


Figure 4.2 Overview of tool ımplementation

4.2 The champion, change agent and team

The person appointed to manage the implementation of the testing tool within the organisation is critical to its successful use: even the best tool in the world will not succeed without an enthusiastic (yet diplomatic) 'champion' or 'change agent'.

The champion may be a different person from the change agent, or there may be only one person with both roles. The champion's role is to provide visible support for the tool within the organisation, and may be played by a high-level manager. The change agent is in charge of the day-to-day progress of the tool uptake while it is being phased into the working practices of the organisation. This may or may not be a full-time job. Another role is that of 'tool custodian' or 'technical visionary'; he or she is responsible for technical tool support, implementing upgrades from the vendor and providing internal help or consultancy in the use of the tool.

The role of 'technical visionary' must not be underestimated for CAST. The technical problems which arise are different than those encountered in other areas of programming and software development, more generally. A technician with a vision of how test automation works, probably based on past experience or some intensive training, will be key to preventing others from abandoning the tool before it has been given a reasonable chance to deliver benefits.

The typical change agent will have been involved in the tool selection process, and is often the same person who was in charge of tool selection.

The qualities needed by the change agent (adapted from Barbara Bouldin's *Agents of Change*) are:

- recent and extensive experience in testing
- progressive rather than reactionary personality
- practical and business orientation, as well as technical background
- highly developed analytical skills.

The team which selected the tool may also be the team which helps to implement it. Ideally it should include people from all the different parts of the organisation that would be expected to use the tool.

Only one tool should be implemented at one time.

It takes time for the effect of the tool to be appreciated, used, and then used effectively. The tool will also change the way the testing is done, and may influence the choice of what to automate next.

4.3 Management commitment

Obviously there is already a level of management commitment to tool support for software testing, because the tool selection process will have been authorised. However, many people fail to recognise that the cost of the tool is more than the cost of the tool. This paradox is true because the tool purchase price is only a small component of the cost of implementing any tool (or indeed a new technique or method) within an organisation.

The cost of a tool is more than the cost of the tool.

This is why management commitment is critical at this point of the tool acquisition process. The change agent must be adequately supported by management in at least two ways: first, visible backing from high-level managers, and secondly, adequate funding and resourcing. The first without the second is classic 'lip service' and is the way to end up with shelfware. The second without the first is seldom successful and makes the change agent's task considerably more difficult, if not impossible.

In order to gain management commitment, the champion or change agent needs to present the business case for the selected tool, summarise the tool selection and evaluation results, and give realistic estimates and plans for the tool implementation process.

The commitment needed from top management is not just a one-off agreement to purchase the tool, but needs to be continual throughout the implementation process. Managers also need to realise that the first thing which happens when a new tool is used for the first time is that productivity will go down, even when the tool is intended to increase productivity. Adequate time must be allowed for learning and 'teething problems', otherwise the tool will be abandoned at its point of least benefit and greatest cost.

Advice on the best way to manage the change process within an organisation is available from consultancies and books.

4.4 The publicity drive

Once you have the management commitment, both verbal and financial, the change agent needs to begin putting in place a continuing and highly visible publicity machine. All those who will eventually be affected need to be informed about the changes.

The first step is simply to raise interest in the new tool, for example by giving internal demonstrations, sending a newsletter, and just going and talking to people about it.

The most important publicity is from the earliest real use of the tool. The benefits gained on a small scale should be widely publicised to increase the desire and motivation to use the tool. 'Testimonials', particularly from converted sceptics, are often more effective than statistics.

4.5 Internal market research

In parallel with the publicity drive, the change agent and the change management team need to do a significant amount of internal market research, talking to the people who are the targeted users of the tool. Find out how the different individuals would want to use the tool, and whether it can meet their needs, either as it is or with some adjustments. The lines of communication set up by interviewing potential tool users can also be used to address the worries and fears about using the tool that contribute to people's resistance to change.

4.6 Pilot project

It is best to try out the tool on a small pilot project first. This ensures that any problems encountered in its use are ironed out when only a small number of people are using it. It also enables you to see how the tool will affect the way you do you testing, and to modify your existing procedures or standards to make best use of the tool.

The pilot project should start by defining a business case for the use of the tool on this project, with measurable success factors.

The use of the testing tool will change your testing procedures in ways which you will probably not expect. To understand this effect, consider the following analogy to purchasing a domestic appliance:

> *Suppose you decided that washing dishes was taking up too much time. After considering other alternatives, such as hiring someone to help with the housework, you decided that the answer was to buy a dishwasher. You made a list of the essential and desirable features of the dishwasher, and started comparing the different offerings against this list. You asked some of your friends about dishwashers, and through their experience became convinced that you needed to add 'quiet operation' to the list of features. Eventually, you purchased and installed a particular machine which best met your requirements. Now, 3 weeks later, you find that the dishwasher has altered your household routine. On the one hand, it had the desired effect and reduced the activities associated with washing up from 40 minutes to 15 minutes after each meal. However, you now find that you spend about 10 minutes unloading the dishwasher every morning. Previously, you put dishes away as you washed them up, so this is effectively an extra job to do. Furthermore, you no longer have the social time of chatting while washing up after a meal, so you feel that you have lost something of value along with gaining the extra time. In short, the dishwasher changed your domestic routine in ways which you did not expect.*

The implementation of a new testing tool will often have an effect which has not been anticipated. For example, using a test execution tool may make debugging more difficult; previously, when testing manually, you knew where you were when something went wrong, which would help you to find the bug. Using the tool, you only know afterwards that something went wrong, and you then have to spend time recreating the context of the bug before you can find it. So there is an extra job to do which you did not have to do before.

You may find that you lose some value in the testing process through the use of the tool as well. For example, when running tests manually the testers would often think of other things to test while they were supposed to be following a script. They could either divert from the script or make a note of new tests to add, but the inspiration for the new tests was a by-product of running the tests. Having a tool run the tests guarantees no diversion from the script, but also does not yield any new tests.

One of the most important things which must be investigated in a pilot project for a test execution tool is the way in which the tests themselves will be organised. The way the test scripts are first stored in the tool, especially if the scripts are created by capturing input, will not be suitable for long-term effective test automation. There are a number of heuristics which have been evolved through experience, which can make test suites many times easier to maintain. At the very least, you are strongly urged to take sufficient time to experiment during the pilot project to discover how to build automated test suites which will be sufficiently easy to maintain in real situations. If not, the investment in the tool may well be lost. It is not uncommon for tools to be abandoned because it would take longer to maintain the tests when the software changes than to rerun them manually.

4.7 Evaluation of pilot

After the pilot project using the new tool is completed, the results are compared to the business case for this project. If the objectives have been met, then the tool has been successful on a small scale and can safely be scaled up. The lessons learned on the pilot project will help to make sure that the next project can gain even greater benefits.

If the objectives have not been met, then either the tool is not suitable or it is not yet being used in a suitable way (assuming that the objectives were not overoptimistic). Decide why the pilot was not successful, and decide the next steps to take. Do not attempt to use the tool on a wider scale if you cannot explain why it has not succeeded on a small scale! The overheads for start-up may be much more significant on a small scale, for example, and may not have been adequately taken into account in the initial business case. It is best to proceed fairly cautiously in scaling up, and to increase tool use incrementally by one project group at a time.

4.8 Planned phased installation

Assuming the pilot project was successful, the use of the tool in the rest of the organisation can now be planned. The success of the pilot needs to be widely publicised, internal training and in-house manuals need to be organised, and the projects to use the tool should be scheduled. The change agent and change management team can act as internal consultants to the new tool users, and can perform a very useful role in co-ordinating the growing body of knowledge about the use of the tool within the organisation.

It is very important to follow through on the tool investment by ensuring that adequate training is given in its use. A tool which is not being used properly will not give the benefits which could potentially be realised. Every tool user should be properly trained by the vendor. It is not enough to send one or two people on a training course, and expect them to 'train the others'. If they are really expected to train the others, then they must be explicitly trained – the others will not learn by osmosis. The internal trainers will need a level of expertise in the tool similar to the vendor's trainers, which only comes with years of experience; they will also need to be excellent presenters and be allowed time to put together an internal course, which typically takes many times the course time (10–25 hours' preparation per course hour). There may be scope for a brief introductory course to get started, followed by more detail after a few months of use, with regular technical updates and expert tips at, say, 6-monthly intervals after that. The cost of training is paid back many times by expert use of the tool, and this is what gives the real payback from the tool investment.

4.9 Evaluation

It is critical to continue to evaluate the results achieved through the use of the tool in all projects, and to compare the results with the original business case for the tool. If the tool is not fully meeting the goals and objectives which were set out for it initially, a strategy for increasing the achievement of the goals (such as extended in-depth training in use of the tool or training in testing techniques) should be investigated.

Once the tool is being used successfully in a number of areas (after 6 months, for example), then you can look ahead to the next area to improve or provide tool support for.

Crib Sheet of Main Terms/Concepts Used

Foundation-Level Software Testing Course

Term	***Description***
Failure	A deviation of the software from its expected delivery or service A failure occurs when software does the 'wrong' thing.
Fault	A manifestation of human error in software, also known as a defect or bug. Faults may be caused by requirements, design or coding errors.
Error	A human action producing an incorrect result. Human error is inevitable in a complex activity.
Coverage	What we use to quantify testing. Defines an objective target for the amount of testing to perform. Measures completeness or thoroughness.
Acceptance Criteria (exit, closure or completion criteria)	Trigger to say: 'we've done enough '. Objective, non-technical for managers. E.g. all tests executed without failure, all faults corrected and re-tested, all outstanding incidents waived.
Expected outcome, expected result	The behaviour predicted by the specification of a component or system under test when a set of inputs are defined.
Baseline	Requirements, specs. Etc. are baselines. They tell us what the software is meant to do.
Goal: locate faults	A successful test is one that locates a fault. Finding faults is good, perceived as improving quality; testers create 'tough' tests, not easy ones.
Re-testing	If a test finds a fault that we get fixed, we repeat the test that found the fault to ensure the fault has been corrected. This is a re-test.
Regression Testing	When we receive software that has had faults fixed, we may execute tests of unchanged software to make sure they haven't been adversely affected by the change. This is regression testing.
Validation	Determination of the correctness of the products of software development with respect to the user needs and requirements. "Did we build the right system?"
Verification	The process of evaluating a system or component to determine whether the products of the given development phase satisfy the conditions imposed at the start of that phase. "Did we build the system right?"
Static testing	Reviews, walkthroughs, inspections of (primarily) documentation, including requirements, designs, code and test plans.
Dynamic testing	Test which involve executing the software. Component, link, system and acceptance tests are all dynamic tests.
Front-loading	Starting test activities early. Static tests of documents, early test case preparation 'tests ' the document on which the cases are based. Requirements, specification and design faults are detected earlier and are much less costly.
V-Model	A model for development and testing that promotes a staged approach to testing where the baselines are the products of earlier development stages.
Component (unit, module, program) testing	To demonstrate that a single program, module or component performs as described in its specification. Black and white box. Usually done by the component's author.
Link testing (integration testing in the small)	To demonstrate that a collection of components interface together as described in the physical design. White box tests of a sub-system or small group of components sharing an interface. Done by a member of the programming team.

Term	*Description*
Functional System Testing	To demonstrate that a whole system performs as described in the design and requirements documentation. Black box, mainly. A sub-system or system tested by a test team or group of independent testers.
Non-Functional System Testing	To demonstrate that the non-functional requirements (e.g. performance, volume, usability, security) are met. Specialist 'functional tests' against particular requirements or test performed by or using tools. A complete, functionally tested system tested by a test team or group of independent testers.
Large scale integration testing	To demonstrate that a new or changed system interfaces correctly with other systems. Black and white box. A collection of interfacing systems tested by Inter-project testers.
User acceptance testing	To satisfy the users that the delivered system meets their requirements. Functional tests only. Users supported by test analysts test an entire system.
Integration strategies	Bottom up – components, sub-system, and then system top down – top level first then next until eventually completing big bang – assemble entire system in one go.
Driver	Code written to execute a component in isolation when its calling program doesn't yet exist.
Stub	Code written to substitute for features not yet written, so calling programs can be tested.
Alpha testing	Tests done by a supplier on early versions of a product.
Beta testing	Tests done by volunteer users on early versions of a product.
Performance testing	Tests to demonstrate that response time and throughput requirements can be met.
Stress testing	Tests to explore the behaviour of the system under extreme loads.
Security testing	Tests to demonstrate (CIA) Confidentiality, Integrity, Availability requirements are met.
Usability testing	Tests to ensure users can operate the system effectively and efficiently without being frustrated.
Storage testing	Test to ensure the system can accommodate planned amounts of data on disk or in memory.
Volume testing	Tests to ensure large tasks (e.g. large end of month jobs) can be run.
Installation testing	Tests to ensure the system can be installed and used correctly (and re-installed, de-installed).
Documentation testing	Tests to demonstrate that user manuals and guides are accurate, complete, consistent and helpful.
Backup and recovery testing	Tests to demonstrate that backup and recovery procedures are effective.
Maintenance Testing	Testing of changes and enhancements to existing systems. Dominated by regression testing.
Black-box tests (Functional)	Tests derived from the baseline document, without regard to the structure of the system or other implementation details. System/Acceptance tests are mainly black box.
White-box tests (Glass-box or structural)	Tests derived from the internal structure (i.e. code) of the system. Component testing is mainly White box.
Test Design (Technique)	The process of selecting test cases. A method used to derive or select test cases.

Crib Sheet of Main Terms/Concepts Used

Foundation-Level Software Testing Course

Term	*Description*
Test Measurement (technique)	A method used to measure test coverage. A technique identifies test coverage items that can be counted during test design and execution.
Equivalence partitioning	Where the inputs (or outputs) of a system are categorised using rules, and there is one test case per rule.
Boundary value analysis	Where values on and just outside the boundaries of equivalence partitions are used as test cases.
Statement testing/coverage	Where each statement in code must be exercised by at least one test.
Decision/branch testing/coverage	Where each decision outcome or branch must be exercises by at least one test.
Error-guessing	Tests derived from knowledge and experience of previous problems. Used as a 'mopping up' approach.
Review	A method of examining documentation or code with a view to detecting, recording and tracking faults, communicating information and improving the development process.
Walkthrough	Aimed at communicating information, a document author leads reviewers through a document to explain the content.
Technical (Peer) review	Aimed at finding faults and achieving consensus on the way forward. Led by a reviewer and involving technical peers.
Inspection	Aimed at finding faults based on pre-defined rules, let by a trained moderator and involving technical peers.
Static analysis	Analysis of code to detect faults, without actually executing the software – normally done using a tool.
Configuration management	A disciplined approach to controlling the creation, organisation, versioning and build of software components.
Incident (observation) report	An unplanned event that (normally) documents a test failure or has an adverse effect on the testing.
Static analysis tool	Used to analyse code and detect poor programming practices or over complex code.
Capture replay tool (test execution tool)	Used to execute prepared tests automatically. Normally have a programmable scripting language.
Dynamic analysis tool	Used to report on the state of the internal variables in code used while the test is still running.
Source code coverage analysis tool	Used to report the statements or decisions covered by a test to help component testers to derive additional tests to meet a coverage target.
ISO 9000	A standard for quality in software testing. Says that testing should be done.
DO178b	A standard for airborne software. Says how much software should be done.
IEEE 829	Standard for Software Test Documentation.
BS7925-1	Standard Glossary of Testing Terms.
BS7925-2	Standard for Software Component Testing.

Test Question Paper 1
(use after slide 50 – Limitations of Testing)

1. If we run the same test twice, is the software twice as tested?

   A Yes

   B No

2. Which of the following is a software failure?

   A The variable 'TODAY' is not initialised in a program, when started.

   B The business analyst failed to document all the requirements in a software requirements specification.

   C The software which performs the premium calculation for an insurance policy produces the wrong result.

3. Which of the following is a software fault?

   A A programmer forgot to write the validation code for a numeric value.

   B Some code that validates a month number reads:

   ```
   IF MONTH < 1 OR MONTH >=12 THEN
      Display "Invalid month"
   ENDIF
   ```

   C A screen rejects "12/01/99" as an "invalid date"

4. Which of the following would you think are 'reliable'?

   A An operating system which failed once a year

   B An operating system which failed once a month.

   C An operating system which failed once a day.

   D All of these could be deemed reliable.

5. Which is probably the most expensive type of fault to correct, late in a project?

   A A faulty requirement

   B A faulty design

   C A faulty program

6. Which of the following affect the cost of testing?

A Degree of risk to be addressed

B The efficiency of the test process

C Level of automation

D Skill of the personnel

E The target quality required

F All of the above.

7. When you ask users to test software, what approach should they take?

A They should scan the user manual looking for situations of interest which they then test, menu option by menu option.

B They sit at the terminal, dreaming up situations randomly visiting all the features of the system to ensure they get broad coverage of the system.

C They work out what business processes are supported by the system, and derive tests that are based on the most important paths through their business process.

D Users cannot test software effectively.

8. Why do we test?

A To ensure that a system does what it is supposed to do

B To assess the quality of a system

C To demonstrate to the user that a system conforms to requirements

D To demonstrate to the users they are getting what they ordered

E To learn what a system does or how it behaves.

F All of the above.

9. Do you think software can ever be fault free to the degree that it is absolutely perfect?

A No

B Yes

10. Why is safety critical software so expensive?

A Because the hardware on which it runs is very expensive.

B Because software engineers are highly paid.

C Because software engineers are very careful so take much longer to write code.

D Because the risk of failure is so high that much more testing has to be done and paid for.

Test Question Paper 1

Answers:

1.	B
2.	C
3.	B
4.	D
5.	A
6.	F
7.	C
8.	F
9.	A
10.	D

Test Question Paper 2
(use after slide 148 – User Acceptance Testing)

1. If 100 bugs were found by users of some production software in the first six months of use, what could you say about the quality of the testing?

   A I think the testing was good.

   B I think the testing was bad..

   C I couldn't say whether it was bad or good.

2. Which of the following statements is the best definition of a test?

   A A controlled exercise having (potentially) several objectives and conducted under controlled conditions.

   B A demonstration intended to show that a system does what it should do.

   C An exercise intended to detect (and eliminate) faults.

   D A controlled experiment intended prove that a system does not do what it should.

3. Which of the following is NOT a test activity?

   A Planning.

   B Preparation.

   C Execution.

   D Check for Completion.

   E Recording.

   F ALL of the above are test activities.

4. What is a sensible goal for testers?

   A To show that a system works.

   B To show that a system does not work.

   C To detect faults in the system.

   D To show that there are no faults left in the system.

5. Which is the best attitude a tester should have to developers and their code?

   A Subservient and co-operative with developers.

   B Aggressive and adversarial to developers.

   C Friendly with developers and trusting of their code.

   D Friendly to developers, but sceptical of their code.

6. Here are some statements about regression testing. Select the one that is false.

A Regression tests are not necessarily separate tests.

B Regression tests are especially important when doing maintenance to a system.

C Effective regression testing is best done using automated tools.

D Regression tests are always run before re-tests.

7. Two of the following three statements are always true for both regression testing and re-testing. Which statement is always true, but only for a re-test?

A We are running a test that we have run before.

B The last time we ran the test it found a fault.

C We have expected results for the test.

8. Three of the four statements below are true for verification activities, but which one is true for validation?

A Can use static testing techniques (reviews, inspections, walk-throughs).

B Usually involves the use of baseline documents.

C Always needs user involvement.

D Checks whether we built the system right.

9. Which of the situations below make systems more difficult to test?

A The requirements/specifications are unclear.

B Testers were not involved in the review of the requirements.

C The software provides little information about its internal state.

D The software is so complex, it is difficult to calculate the expected results correctly.

E All of the above.

10. Which of the following statements about the stages of testing is false?

A Test stages occur at different times throughout the development life cycle.

B We concentrate on finding different faults in different stages.

C We test in different stages because the difficulty varies.

D Sometimes different people are responsible for different stages of testing.

Test Question Paper 2

Answers:

1.	C
2.	A
3.	F
4.	C
5.	D
6.	D
7.	B
8.	C
9.	E
10.	C

Test Question Paper 3
(use after slide 215 – Integration test Coverage)

1. Which of the following is not a test stage?

   A Component testing

   B Integration testing (in the small and in the large)

   C Functional system testing

   D Dynamic testing

   E Non-functional testing

   F User Acceptance testing

2. Three of the statements about formal component testing are true. Which statement is false?

   A Done entirely with white box techniques.

   B Has a test plan.

   C Is repeatable.

   D The faults are logged.

3. Which of the following is true of the Testing Standard (BS 7925-2)?

   A It covers the selection of test design and measurement techniques.

   B It is intended to be auditable.

   C It covers dynamic and static testing.

   D It is a standard for system and acceptance testing.

4. In the context of test techniques, which one of the following statements describing a model is false?

   A A test model is the best test you can do.

   B A model may help us to analyse code for the purpose of deriving test conditions.

   C A model may help us to document and diagram the code or requirements.

   D A model could be used to test a specification or requirement.

5. Which of these four statements about link testing is false?

   A Link tests explore direct and indirect interfaces.

   B Link tests identify inconsistencies between components.

   C Link testing is a white-box activity.

D Link tests are usually done by someone outside the development team.

6. Which of the following is not an interface?

   A A statement passing control to another program.

   B Parameters passed from one program to another program.

   C A program writing to a file that is only used by that same program.

   D Global variables defined at the time of transfer.

7. What is the most important reason for doing a User Acceptance Test?

   A To find the bugs that haven't been found in system testing.

   B To demonstrate that the system fits the way users want to work.

   C It is the only test that is planned and performed by the user.

   D To give users an opportunity to learn the system.

8. Which statement is true?

   A Alpha and beta testing take place after delivery to customers.

   B System testing usually follows User Acceptance testing.

   C User Acceptance testing usually follows System testing.

   D Contract acceptance testing doesn't need acceptance criteria.

9. Three out of the four statements below are true about large scale integration. Which is the statement that is false?

   A If a system does not interface to any other systems, large scale integration testing is not required.

   B An inventory of the interfaces between systems is key to large scale integration testing.

   C This testing should begin prior to system testing.

   D Large scale integration testing is usually based on business transactions.

Test Question Paper 3

Answers:

1.	D
2.	A
3.	B
4.	A
5.	D
6.	C
7.	B
8.	C
9.	C

Test Question Paper 4
(use after slide 318 – From Examples of Traps 2)

1 Which statement describes the key difference between black and white box testing?

A Who performs it.

B Where it occurs in the life cycle.

C Whether you can 'see' the code.

D Whether you have a specification.

2. Which of the following statements about statement and branch testing is true:

A These are static test techniques.

B When we talk about statement and branch testing, there is an implication that we will measure the coverage.

C Statement and branch testing are usually done by the testers, not the developers.

D When we run tests, we say the paths have been "sensitised".

3. In general, each test technique 'models' the software under test to highlight the code in such a way that test cases can be constructed to find likely faults. Is the following statement True or False? The 'model' for Error Guessing is the experience of where errors have existed in software in the past.

A True.

B False.

4. Performance testing is a once-off event, done at the end of the development? True or False?

A True.

B False.

5. Which of the following statements about security testing is TRUE?

A Security testing must always be done at the end of the development cycle.

B Security testing has only become an issue since the internet.

C You need to test security features both positively and negatively.

D Security testing is best done by system administrators.

6. Which of the following statements about usability is FALSE?

A If the usability requirements are fuzzy, it will be difficult to derive tests that have objective test results.

B Usability is about making a system that allows users to do their job efficiently and effectively.

C Usability is concerned with the entire user experience.

D Usability is less important than it used to be.

7. Which of the following is a true statement about black box testing?

A Black box and functional test techniques are different.

B Needs test tools to use the technique.

C Does not require a knowledge of the internals of the software.

D Black box tests do not use the baseline.

8. Which of the following is a true statement about white box testing?

A White box, glass box and structural test techniques are different.

B To use white box test techniques effectively, you need tools.

C Does not require a knowledge of the internals of the software.

D Black box tests do not use the baseline.

Test Question Paper 4

Answers:

1. C
2. B
3. B
4. B
5. C
6. D
7. C
8. B.