ISEB Practitioner Course Notes
Based On Certificate Syllabus

Mark Robinson – 11 June 2005

mark.robinson@sioux.nl

New Features For This Version:

- **Too many changes to mention, throughout document. Thanks Bryan and Mario! (Thanks also to Erik for the encouragement!)**

Thanks to fellow contributors:

Bryan Bakker	– absolutely tons of contributions throughout document.
Steven Deneir	– starting the ISEBCertification Yahoo! group and exam tips added to Appendix B.
Mario Peeters	– an unfeasibly large number of improvement suggestions, contributions and V-model table (Appendix A) and many diagrams.
Hans Schotanus	– very many contributions throughout document, including Risk categorisation diagrams.

Thanks to all these chaps for their help. Occasionally their work is directly credited in this document, but most of the time the references are made directly to the source documents. So, though it is not obvious to third parties, these guys have put in *a lot* of work. Thank you very much: Bryan, Steven, Mario and Hans: it was real team work.

Thanks also to Erik van Veenendaal who taught the ISEB Practitioner course we all took *very well* and wrote "The Testing Practitioner" book that we use and extensively quote here. Erik – without you we would have found passing the exam *much* harder… and this document would be much thinner!

0 Introduction To This Document

0.1 Author's Notes

- These notes provide answers to the points raised in the syllabus. They are **not** intended to be a full explanation of all topics covered on the ISEB Practitioner Course but hopefully address most of the points in the Syllabus. (The course sometimes goes beyond the syllabus since the goal is not just to pass an exam but learn techniques that can usefully be applied in the real world.)
- Text in *italics* is direct from the syllabus (sometimes the format is changed slightly, e.g. use of bullet points). Non-italic text are my answers and comments.
- The references to the appropriate pages are shown as follows:
  - CN/A-03/1-2: pages 1 & 2 of module A-03 of course notes.
  - CN/TMR/41 page 41 of Test Management Reader of course notes
  - TP/45: page 45 of "The Testing Practitioner".
  - MARK: no notes could be found, I have made my own.
- Thanks to http://www.wordsmith.org/anagram for providing anagram suggestions (if you do not like the ones I chose, you can replace them with your own using this site). Anagrams are rendered in **GREEN BOLD CAPITALS** (hopefully not too garish).
- **Disclaimer:** this document has been provided for free as a service by myself and others mentioned in the preceding pages. It is not guaranteed in any way, e.g. free from errors or complete. Furthermore, it is not intended to break copyright or in any way infringe upon ISEB. Hopefully, rather, it encourages people to take the course and helps them to pass the exam.

0.2 Objectives of the Practitioner Certificate Qualification

(This section is copied directly from the syllabus.)
The successful candidate should be able to:

- *Describe strategies for the software testing of both new development and the maintenance of existing systems.*
- *Describe different strategies for the testing for both complete life cycles and individual phases.*
- *Plan the testing needed at any level from component to user acceptance testing and document it in compliance with IEEE Std. 829-1998.*
- *Analyse risks and use the results to prioritise the testing.*
- *Specify and design test cases.*
- *Define requirements for an appropriate test environment.*
- *Run tests using defined test procedures.*
- *Log, analyse and report incidents.*
- *Interact effectively with others such as users, developers and managers.*
- *Participate in reviews.*
- *Select and implement tools to support testing activities.*
- *Assess testing and development activities for possible improvement.*

0.3 Your Contributions To This Document

If you would like to add to the document, please write out (or draw) in full what you want to add so that I can just copy and paste it. Please make sure that all I need to do is Ctrl-C, Ctrl-V. Anything else is hard work… ☺ Please use same formats (e.g. paragraphs, bullet points and justification). I may change it before adding it, or not add it at all. Please mention **paragraph numbers** (which are fixed) rather than page numbers (which vary between versions).

If you have any ideas for improvement, if you spot defects or if you have any other constructive criticism, please let me know.

If you think this document is useful I would be very glad to hear from you (nice to know it was worth the effort). You may distribute it to others (but new versions will be made exclusively by me).

0.4 Abbreviations

Abbreviation	Explanation
BS	British Standard
CCB	Configuration control Board
CE	Cause Effect Graphing
CI	Configuration Item
DSDM	Dynamic System Development Method
ECT	Elementary Comparison Test
ET	Exploratory Testing
FMEA	Failure Mode and Effect Analysis
IEEE	Institute of electrical and Electronics Engineers
ISEB	Information Systems Examinations Board
PBR	Perspective Based Reading
PM	Project Manager
RAD	Rapid Application Development
ROI	Return On Investment
RPN	Risk Priority Number
SUMI	Software Usability Measurement Inventory

1 Introduction

1.1 Review of Foundation Certificate Syllabus

Introduce the issues, philosophy and key points of software testing as covered in material based on the current syllabus of the ISEB Foundation Certificate in Software Testing.

Error: mistake made by human.
Fault: result of an error (may go unnoticed).
Failure: visible effect of a fault. TP/13

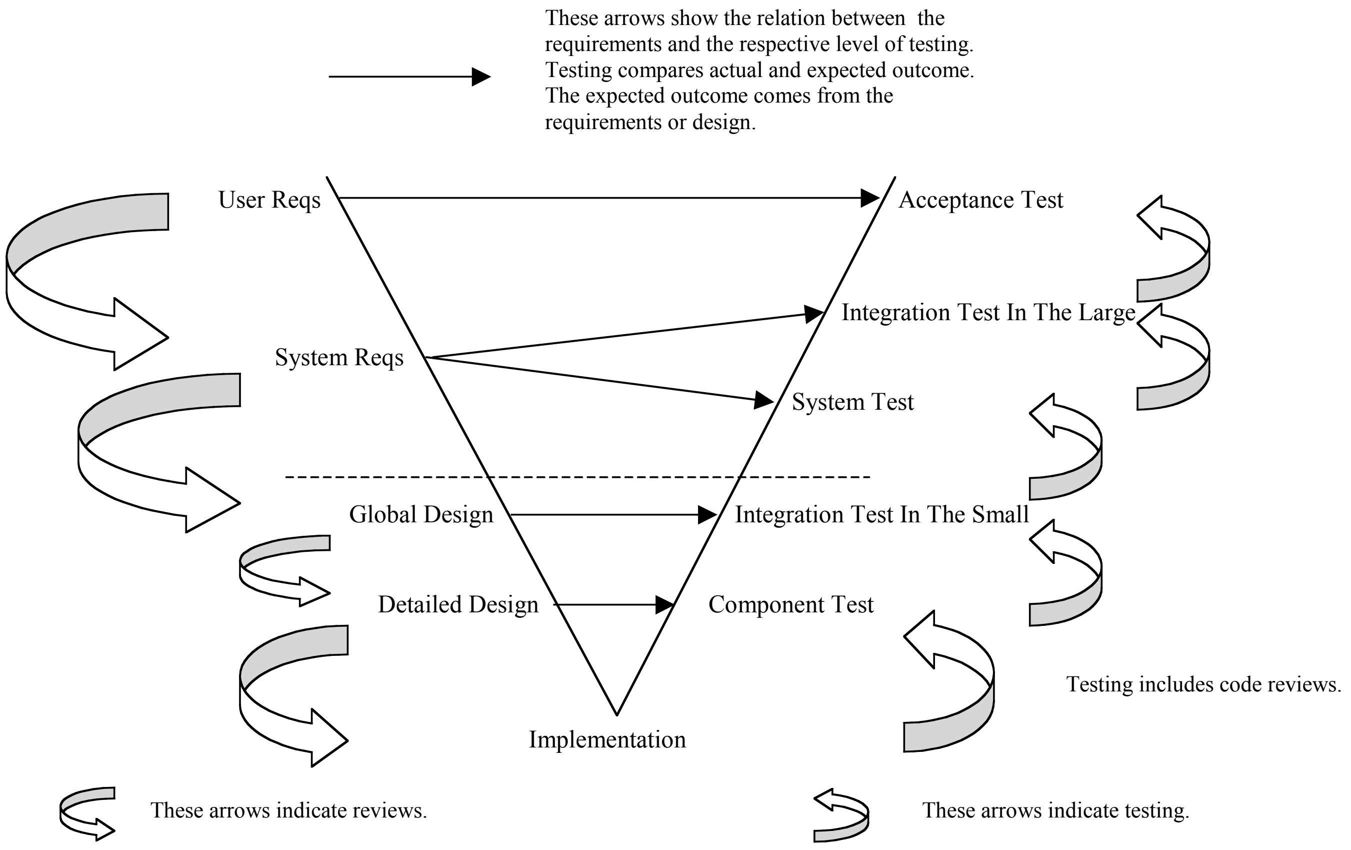
Provide an introduction to the ISEB Practitioner Certificate as the next stage in career development.

1.2 Testing in the Life Cycle

Describe how testing fits into the different life cycles: sequential (e.g. waterfall, V-model), iterative (pre-planned incremental delivery and evolutionary delivery) and RAD (e.g. DSDM).

Waterfall: project is developed in phases, e.g. Requirements, Design, Coding, Integration, Testing. Testing is done only at end. MARK

V-Model: testing is performed throughout the life cycle. Static testing (reviewing of documents, by Inspection, Walkthrough or Peer (Technical) Review) is performed on documents early in life cycle (e.g. on User Requirements, System Requirements, Global Design, Detailed Design and Code). Testing later in life cycle is also performed in five phases, each complementary to the early phases (Component Testing, Integration in the Small Testing, System Testing, Integration in the Large and Acceptance Testing). TP/23-29


These arrows show the relation between the requirements and the respective level of testing. Testing compares actual and expected outcome. The expected outcome comes from the requirements or design.
User Reqs
Acceptance Test
Integration Test In The Large
System Reqs
System Test
Global Design
Integration Test In The Small
Detailed Design
Component Test
Testing includes code reviews.
Implementation
These arrows indicate reviews.
These arrows indicate testing.

<u>Pre-Planned Incremental Delivery:</u> similar to waterfall but broken up over a number of increments. Requirements are broadly outlined for all increments at start of project. A software release occurs at the end of each increment in which two tests occur: thorough testing of new functionality and regression testing (partial re-testing of previous version). The amount (of regression testing) may grow or may need to be refined at each increment. MARK

<u>Evolutionary Delivery:</u> similar to pre-planned incremental delivery but requirements are developed separately for each increment, often during the life time of the previous increment. TP/29

<u>RAD (Rapid Application Development):</u> Similar to Evolutionary but each increment is over a much shorter time span, typically two weeks. Requirements may not formally be specified. MARK

Four cornerstones of TMap: Life Cycle, Techniques, Infrastructure, Organisation TP/31

Introduce the concept of interfaces between test process and other processes, such as project management, configuration management and change management, software development, technical support and technical writing.

The test process does not exist in isolation in a project but rather works together with other processes. For example:

Project Management: without a project plan there is no basis for testing. The project plan identifies stakeholders, defines the life cycle, defines the deliverables, defines milestones, provides estimates upon which scheduling is based, establishes risk management which identifies where test effort should be focused, tracks effort, schedule and risk and stores measurement data. PM are ultimately responsible for all deliverables and need to manage and communicate changes. CN/A-03/1, TP/367

Requirements Management: without managed requirements (formal) test designs are almost impossible. Requirements are documented and reviewed. Requirements are basis for design, code, deliverables and testing. Changes are permitted only via a documented procedure. Requirements Traceability must exist (horizontal and vertical throughout V-model). May make use of supporting tools. CN/A-03/1-2

Configuration Management: without CM it is impossible to know what has been changed and what needs to be tested. It is a set of procedures for tracking and documenting products through their life cycle, to ensure all changes are recorded and the current state of the products is known and reproducible. It is concerned with identification, control, auditing and status accounting. CM is necessary for managing changes during development, test and delivery. Also used for testware and test environment. CN/A-03/1-2, TP/367

Change Management: see Configuration Management above. Or Structured Configuration Management, below. MARK

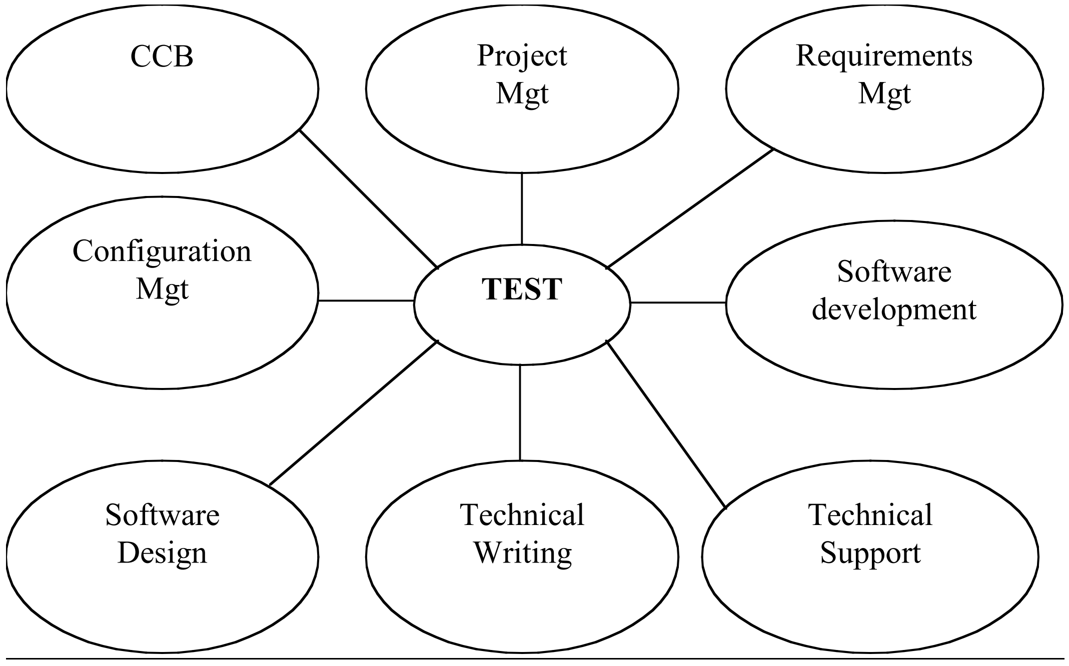
Structured Configuration Management: prepared for each project. Everything (software, documents) placed under CM. Change Requests and Problem Reports are initiated, recorded, reviewed, approved (prioritised) and tracked. A Configuration Control Board (CCB) exists and a member of the test team should be on it. This is to inform the CCB of estimates (including risks) for each change and so that testers know what changes are accepted. Products are created from the software baseline library and their release is controlled according to a documented procedure. The status of Configuration Items (CIs) is known. Software baseline audits are conducted. CN/A-03/3, TP/367

Software Development: Life cycle has a big impact on testing. Phased releases have cumulative test cases and more regression. A small change can mean a lot of testing, re-testing and regression testing. Development (effort) and testing (effort) are related but the relationship is not always linear (directly proportional). Also agree release time, content and procedures. Interact regarding faults and change management. CN/A-03/2, TP/367

Technical Support: testers can ask TS questions regarding live faults, quick fixes and (potential) workarounds. Testers can improve their tests by finding out from TS about faults found in the real world. Testers can help TS by carrying out priority regression testing. TP/367

Technical Writing: technical authors can gain information on the system operation and performance by working with the test team. The test team can validate any products, e.g. user guides. They can help TAs (Technical Authors) understand how the product will be used. TP/367

Software Design: During test case design, numerous queries will arise from a technical design nature, which will require confirmation. Including a designer in the review process for testing is useful as it can often highlight areas that have been missed or are considered to have insufficient coverage from a technical design viewpoint.



TP 366

Show how early test planning may be balanced with the later test execution using the 'V' model. Explain the differences between verification and validation, and that in the early life cycle phases these are typically achieved using reviews. Highlight the requirement for change management and configuration management in the software engineering process.

At each stage in the early life of the V-model tests can be planned which are later executed in the later life of the V-model. Furthermore, testing can be begun earlier in the life cycle:

Early Life – Testing	**Early Life – Planning**	**Late Life - Execution**
Review acceptance criteria. (Walkthrough)	Acceptance criteria => define acceptance tests.	Execute acceptance tests.
Review requirements document. (Walkthrough)	Business requirements => define system tests.	Execute system tests.
Review design document. (Technical)	Overall design => define integration tests.	Execute integration tests.
Review detailed designs. (Technical)	Detailed design => define component tests.	Execute component tests.

An Inspection is useful in each of these phases before the document goes to the next phase.

Validation: "Is this the right specification?"
Verification: "Is the system correct to specification?"
(Both from BS7925-1 1998) TP/14

Or:
Validation: "Are we making the right product?"
Verification: "Are we making the product right?"
MARK

The walkthrough of a working document is part of the validation process. TP/134

Change Management and Configuration Management are detailed above.

Provide a definition of the test phases: component testing, component integration testing, functional system testing, non-functional system testing, system integration testing and acceptance testing. Explain that each test phase has the following characteristics: objective, scope, entry and exit criteria, test deliverables, applicable test techniques, metrics, test tools and applicable testing standards.

See Appendix A: V-Model Table for further details.

Component Testing

- Planned early with detailed designs before coding.
- Run before system testing.
- **Based on detailed design specs.**
- **Based on internal workings of components and at extreme/failure conditions.**
- May be carried out by testers or developers.
- May require non-functional as well as functional tests.
- **Will use both white box and black box testing.**
- May require test tools.

Component Integration Testing (Integration "in the Small")

- Planned and designed early, with system designs.
- Run late but before system testing.
- **Based on detailed design specs.**
- **Based on *links* between components and at extreme/failure conditions.**
- Carried out by testers or developers.
- Functional and non-functional tests.
- May require tools.

System Testing

- Planned and designed early.
- Run late but before acceptance testing.
- **Based on acceptance criteria and requirements.**

- **Based on whole system.**
- Carried out by testers or developers.
- Functional and non-functional tests.
- **Mainly (solely) black box.**
- May require tools.

System Integration Testing (Integration "in the Large")

- Planned and designed early.
- Run late but before acceptance testing.
- **Based on acceptance criteria and requirements.**
- **Based on how whole system interacts with others in its environment: interoperability.**
- Carried out by testers.
- Functional and non-functional tests.
- **Black box.**
- May require tools.

Acceptance Testing

- Planned and designed early.
- Run late (last) in life cycle.
- **Based on acceptance criteria and business view of system.**
- May be contractual
- **Tested by end users.**
- Functional and non-functional tests.
- **Black box.**
- May require tools.

TP/26-29

Acceptance testing - Test techniques

- Equivalence partitioning
- Sometimes Cause/Effect(CE) graphing or Elementary Comparison Test (ECT)
- Use cases / Process Cycle Test
- Non-systematic techniques (Exploratory Testing(ET))

Typical non-functional attributes

- Usability
- Sometimes Performance.

CN/C-15/1

For a more complete list of test techniques at different levels, see Appendix A: V-Model Table.

Each test phase has the following characteristics: Objective, Scope, Entry and Exit criteria, Deliverables, Techniques, Metrics, Tools and Standards. (Copied from syllabus, above.) OSEEDTMTS anagram of **MODE TESTS**.

Explain how the development process can influence the testing process, for example the object-oriented development paradigm where development makes testing more difficult because of information hiding.

As development process differs, so the risks will change. For example, in an evolutionary life cycle, the V-model can be applied in miniature (can build up processes from informal to formal). TP/29

Explain the differences between testing, retesting and regression testing.
Testing: "demonstrating that a system is fit for purpose" TP/17 or "the process of exercising software to verify that it satisfies specified requirements and to detect errors." TP/18

Retesting: "to discover whether or not a previously reported fault has been fixed" MARK

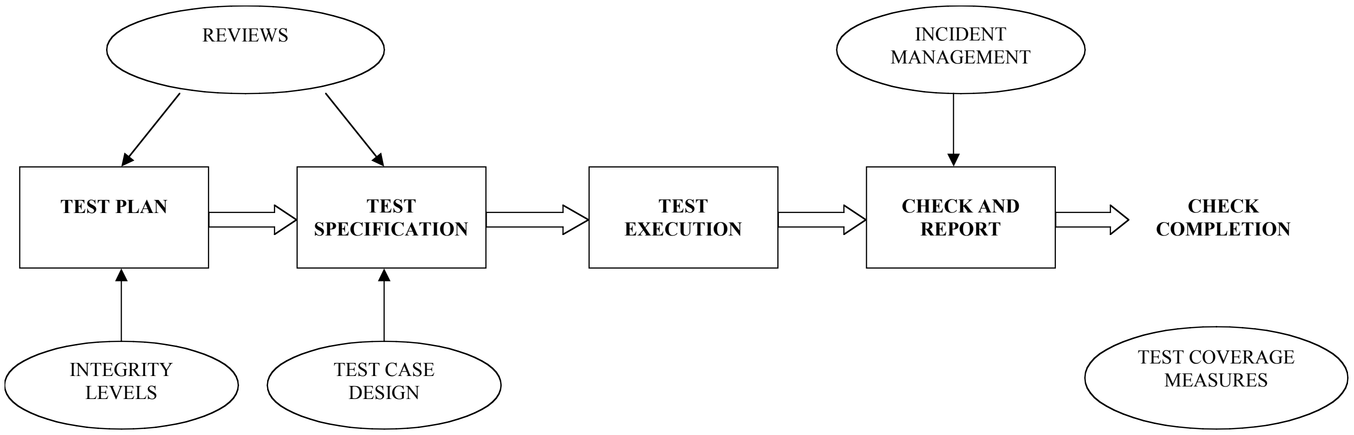
Regression Testing: "testing a representative part of a system to verify that the basic functionality works, often used when releasing a patch or in incremental delivery." MARK

2 Test Process

2.1 Generic Test Process

Introduce the generic test process as comprising the following activities:

- *Test planning.*
- *Test specification.*
- *Test execution.*
- *Test checking and recording.*
- *Checking for test completion.*



TP 46

PSERC, see "3.3 *Test Estimation*" TP/46

Test **p**lan: integrity levels, reviews.
Test **s**pecification: test case design, reviews.
Test **e**xecution: <no details>.
Test checking and **r**ecording: incident management.
Checking for test **c**ompletion: Test coverage measures. TP/46

Indicate how this generic test process fits within the software development process.

This test process can be applied in each phase of the later life of the V-model (e.g. component testing, integration testing, system testing and acceptance testing). MARK

2.2 Test Planning

This topic is covered in detail in section 3.

Section 3 is "Test Management".

2.3 Test Specification

Explain that the choice of test specification techniques should be dependent on

- *the <u>risks</u> to be mitigated,*
- *the <u>knowledge of the software</u> under test and*
- *the <u>source documents</u>*

and that the choice should be <u>justified</u> in the test plan/strategy.

Risks are covered in detail under "4 Testing and Risk".

Explain the role of BS 7925-2:1998 as a source of definitions of test specification (test case design) techniques. Explain that test specification comprises three stages.

- *First to identify the test coverage items (analysis),*
- *Second to create test cases that exercise the identified test coverage items (design) and*
- *Third, organise the test procedure/script and test environment from the test cases (build).*

Prioritisation criteria identified within risk analysis and test planning may be applied at both the analysis and design stages.

BS 7925-2:1998 contains a list of definitions of test techniques that are commonly used throughout the ISEB Practitioner course, e.g. black box testing. Specific techniques are shown in the table below.

Characteristics	Static	Dynamic	Black Box	White Box	Formal	Informal	No formal coverage measure	Formal coverage measure
Techniques								
Equivalence Partitioning		**X**	**X**		**X**			**X**
Ad Hoc Testing		**X**	**X**	**X**		**X**	**X**	
Branch (Decision) Coverage		**X**		**X**	**X**			**X**
State Transition		**X**	**X**		**X**			**X**
Walkthrough	**X**		**X**	**X**	**X**		**X**	
Cause/Effect Graphing		**X**	**X**		**X**			**X**
Complexity Analysis	**X**			**X**	**X**		**X**	
Random Testing		**X**	**X**		**X**		**X**	
Statement Coverage		**X**		**X**	**X**			**X**
Data Flow Testing	**X**	**X**		**X**	**X**			**X**

Syntax Testing		**X**	**X**		**X**		**X**	
Inspection	**X**		**X**	**X**	**X**		**X**	
Error Guessing		**X**	**X**	**X**		**X**	**X**	
Boundary Value Analysis		**X**	**X**		**X**			**X**
Informal Review	**X**		**X**	**X**		**X**	**X**	

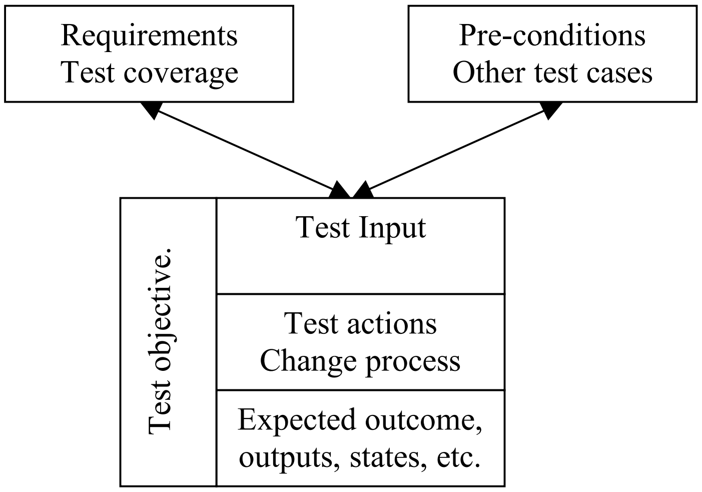
CN/A-Handouts/3

Examples (of three stages of test specification)
Analysis: situations, conditions, attributes, state transitions.
Design: test design specification, test case specification.
Build: may also include organising test data and defining test execution scenario. CN/A-04/7

Explain that BS-7925-2:1998 is one source of definitions of test case design techniques, some of which may be used for component testing, component integration testing, functional system testing, non-functional system testing, system integration testing and acceptance testing. Explain that test cases may be generated from an analysis of business scenarios and/or use cases.

So, BS-7925-2:1998 contains many definitions of test case design techniques which are used in all test phases.

Describe the requirement for each test case in the test specification to include the test input(s), expected outcome, the rationale for the test (i.e. which test coverage items are being exercised by this test) and any test case-specific pre-requisites, such as the initial state of the software and its environment.



CN/A-04/8

Explain that to generate the expected outcome and to perform the analysis stage, knowledge of the specification for the software under test is required. In some situations this knowledge may not be formally *specified. Therefore it is sometimes necessary to identify alternative sources, such as technical and/or business knowledge. RAD is a particular example of where the requirements may not be formally specified. If expected results cannot be generated then testing cannot be performed.*

To specify the tests, the following are needed:

- Functional specification.
- Knowledge of the software specification.
- If no documentation is available, access to business and technical experts.

The test specs need to be reviewed for testability.

The expected outcome needs to be identified, otherwise no testing can take place. CN/A-04/7

Explain that the test outcome must include not only outputs but the final state of the software under test and its environment.

(The comments below are **related** to the paragraph above but not a specific explanation: see point 7.)

Use case test template could consist of the following fields:

1. Test case identifier.
2. Software to be tested and version.
3. Goal of test.
4. Description (including Context and Actors).
5. Preconditions.
6. Description of main flow.
7. Expected results (outputs including final state).
8. (Alternative flows and expected results.)
9. Failure conditions (for usability testing, e.g. instructions not clear).
10. Additional attention points (e.g. note response times.)

TP/245

Describe the non-functional attributes that may have been tested (including reliability, memory management, security, recovery, disaster recovery, volume, performance, stress, usability, maintainability, procedure, configuration, portability, installability, interoperability, compatibility and conversion). The non-functional attributes to be tested should be in the specification of the software under test.

Non-functional attributes (or quality characteristics) are:

- **F**unctionality.
  - Suitability.
  - Accuracy.
  - Interoperability.
  - Security.
- **R**eliability.
  - Maturity.

    - Fault-tolerance.
    - Recoverability.
- **U**sability.
    - Understandability.
    - Learnability.
    - Operability.
    - Attractiveness.
- **E**fficiency.
    - Time behaviour.
    - Resource utilisation.
- **M**aintainability.
    - Analysability.
    - Changeability.
    - Stability.
    - Testability.
- **P**ortability.
    - Adaptability.
    - Installability.
    - Co-existence.
    - Replaceability.

Memory aid: **PERFUMe**.

Note that "Compliance" can be added to each list, e.g. complying to standards relating to Functionality, etc. TP/381

Explain that reviews/inspections, static analysis and dynamic analysis are also used to detect faults and so may be specified as required tests to be performed on the software under test, where the software under test may be requirements specifications, design specifications and code (and test plans).

Additional information:

For static and dynamic lists of tests, see previous table. Reviews are all static tests and may be performed on code or documentation. Review types for documents are:

1. Walkthroughs.
2. Peer (technical).
3. Inspections.
4. Informal.

MARK

2.4 Test Execution

2.4.1 Preparation for Test Execution

Explain that the following are the pre-requisites for test-execution:

- *Test procedure and / or test script.*
- *Identification and establishment of the test environment to include room, desks, trained people, hardware, software, tools, peripherals, communications means, data set-up and security.*
- *That all those who will be responsible for the creation and maintenance of the test environment are identified and available.*
- *That support activities such as configuration management and incident management are in place.*
- *Verification that the test environment is complete and works correctly, i.e. it will correctly demonstrate both test passes and test failures and it is a satisfactory replica of the target environment.*

Pre-requisite summary: procedure, environment, personnel, support, test environment verification. MARK

2.4.2 Executing the Tests

Explain the importance of following test procedures, to ensure that there is complete auditability of the tests and that there is repeatability for retest or regression test. A formal means should also be provided for testers to suggest additional tests that they may like to add.

If someone does not follow the test procedure and misses a fault, then the cost of repairing it will increase when it is discovered during a later phase or when the product is in the field. MARK

Explain that to ensure confidence in the tests, it is sometimes necessary to get the potential owner of the developed software, or someone acting for them, to witness the testing.

The customer may not have the knowledge fully to understand the testing during an earlier phase, e.g. component, so explanation will be needed. Otherwise this participation should be kept to a later phase, e.g. acceptance testing. MARK

Involving the customer during tests ensures that the tests executed are verifying what the customer sees as being important. The customer can also indicate what are the most common problems that were present in previous projects, as such he can indicate what pieces of the system should be tested more rigidly. MARIO

2.5 Test Checking and Recording

2.5.1 Test Checking

Explain that this is a comparison of actual and expected outcomes and that the expected outcome must be generated prior to test execution.

So, say what should happen and then see whether that is the case.

Explain that if any discrepancy between actual and expected outcomes is observed then it shall be logged and analysed to establish where the fault, if any, lies (e.g. fault in the testing, fault in the software under test) and where in the test process we should return to either fix the fault in the testing or test the fix made to the software. Ideally, data to support a subsequent root cause analysis should be recorded.

Faults could be:

- Test environment.
- Computer itself (other software).
- Error during software installation.
- Tester not following instructions correctly.
- Test instructions not correct.
- Software error.

MARK

2.5.2 Test Recording

Explain that every test case is logged in the test record for the purpose of:

- *auditing the testing and*
- *recording test coverage measures for subsequent checking against test completion criteria.*

For example, one of the exit criteria might be 90% branch coverage, so the amount of coverage needs to be recorded.

Another example is 100% of critical tests, 80% major and 50% minor. MARK

Need to record the software tested and its version number. TP/21

Describe the contents of the test record based on those required for component testing described in BS 7925-2:1998 and explain how test records are used in the other phases of testing.

<u>Component Test Recording</u>
The test records for each test case shall unambiguously record the identities and versions of the component under test and the test specification. The actual outcome shall be recorded. It shall be possible to establish that all specified testing activities have been carried out by reference to the test records.

The actual outcome shall be compared against the expected outcome. Any discrepancy found shall be logged and analysed in order to establish where the error lies and the earliest test activity that should be repeated in order to remove the discrepancy in the test specification or verify the removal of the fault in the component.

The test coverage levels achieved for those measures specified as test completion criteria shall be recorded.

Standard for Software Component Testing, Working Draft 3.4 (Grey section 7, second folder)

Test records can be used in other phases:

- To help localise a fault found (reference can be made to earlier tests).
- To help identify the value of testing at each phase by collecting metrics.
- (To help identify root causes of faults.)
- To identify risks, so that later testing can focus on particularly error prone areas.

MARK

2.6 Checking for Test Completion

Explain that before exiting the test process a check against the test completion criteria is mandatory and present examples of test completion criteria for each of the test phases. If the criteria are not met, normally additional tests shall be required. Alternatively, the test plan may be revised to permit the relaxation (or strengthening) of test completion criteria. Any changes to the test completion criteria must be documented, ideally having first identified the associated risk and agreed the changes with the customer.

All exit criteria for each phase will be identified in each phase test plan. They will be determined by a risk analysis. TP/46

<u>Completion Criteria Examples For Printer Software:</u>

- Component Testing: when software is running in a test environment (so not on a printer) and a test stub is used to simulate the printer. An example would be: 95% of test cases are exercised, zero critical defects and five or less serious defects.
- Integration Testing: when the various components are run together, in this case possibly a user interface and print manager software. Completion criteria could be no memory leaks or performance loss (needs to be measured – a specific tool would be of use).
- System Testing: running on a printer. Completion criteria could be no critical defects observed after 100 000 printouts (following established test cases).
- Acceptance Testing: could be run by end users on a printer. Completion criteria could be established against usability criteria, e.g. “ease of use, 3/5 or better”.

MARK

Explain that if all test completion criteria are met then the software is released, e.g. to the next phase of testing.

How do completion criteria help?
1. Helps focus testing on the test objectives
2. Allows an end position to be identified
3. Allows progress towards the end position to be measured and monitored
4. Prevents “wrong” decisions being made under pressure
5. Tell you when you have finished the testing as agreed during the planning phase
6. Note: They should be specified per test level/phase

CN/A-11/5

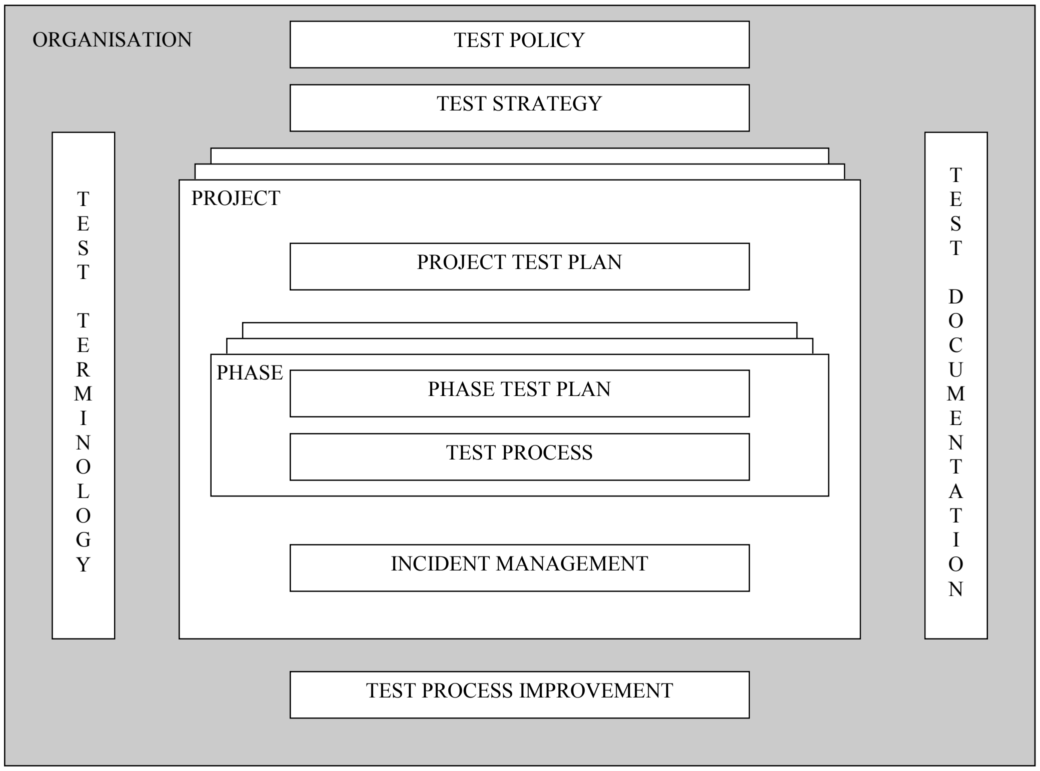
3 Test Management

3.1 Test Management Documentation

Provide a description of the following documents:

Test Policy	*a document characterising the organisation's philosophy towards software testing.*
Test Strategy	*a high-level document defining the test phases to be performed and the testing within those phases for a programme (one or more projects).*
Project Test Plan	*a document defining the test phases to be performed and the testing within those phases for a particular project.*
Phase Test Plan	*a document providing detailed requirements for performing testing within a phase e.g. component test plan, integration test plan.*

For this syllabus, the above four documentation types are used and treated as individual documents, however some organisations may amalgamate and split documents to form different sets which, in total, should contain the same information.



TP/44

Note that Incident Management is not per phase but per project. This is because here it means **management**, not recording and handling (fixing) of incidents. So here it implies, e.g. a CCB (Change Control Board).
MARK.

3.1.1 Test Policy

Explain that an organisational approach to testing starts with a test policy, which is normally developed by the IT department (or equivalent), but represents the philosophy of the whole organisation.

Explain that a test policy is typically a short, high-level document that comprises:

- *a definition of testing (e.g. "Checking that the software solves a business problem."),*
- *the testing process (e.g. "Development and execution of a test plan in accordance with departmental procedures and user requirements."),*
- *the evaluation of testing (e.g. "Measurement of the cost of faults detected after release."),*
- *quality levels to be achieved (e.g. "No more than one high severity fault per 1000 lines of delivered code to be found in the first six months of operation.") and*
- *the organisational approach to test process improvement (e.g. "Post-project reviews will be performed after each project.")*

How can these five points (definition, process, evaluation, quality and improvement) be remembered? DPEQI: Do, Perchance, English Queue Intensely? MARK

Another policy definition:
- definition of testing
- definition of the test process (level of independence)
- strategic view towards the test profession
- evaluation of testing (performance indicators)
- quality levels to be achieved (important quality attributes)
- approach to test process improvement (objectives)

CN/A-04/2

Explain that the testing policy applies to both new development and maintenance testing activities.

3.1.2 Test Strategy

Explain that the test plan should be based on the test policy.

Explain that the scope of the test strategy (as used within this syllabus) covers the generic test requirements for an organisation or programme (one or more projects).

Describe a test strategy as a document that addresses the risks and presents a process for mitigating those risks in line with the testing policy, explicitly showing the link between the risks and the testing. A test strategy will thus typically comprise two main parts:

- *the risks that are to be addressed by the testing of the software and*
- *the particular testing which will be used to address the identified risks.*

100% testing of software is impossible which is why a test strategy is developed, determined by risk analysis. The aim is to focus the most testing effort on the areas where defects are most likely to be found, or where the defects are most likely to have the highest impact (or both). TP/34

Explain that the testing described in a typical test strategy will include a description of the test phases that are to be used. For each of these phases, the following shall be described at a high level (along with the rationale for its selection):

- *the **entry and exit** criteria for each test phase;*
- *the **approach** to testing, such as top-down, bottom-up, priority-driven;*
- *the **test case design techniques** to be used;*
- *the test **completion criteria**;*
- *the degree of test **independence**;*
- *any **standards** that must be complied with;*
- *the **environment** in which software tests will be executed;*
- *the approach to test **automation**;*
- *the degree of **reuse** of software;*
- *the approach to **retesting and regression** testing;*
- *the test **process** that shall be used, including test **deliverables**, such as test reports;*
- *measures / **metrics** to be captured;*
- *the approach to **incident management** to be used.*

A CAD PRIME TIRES or **ERIC, A MAD PRIEST** (note Eric with a 'c'): E(ntry & Exit), R(egression & Retesting), I(ndependance), C(ompletion Criteria), A(pproach), M(etrics), A(utomation), D(eliverables), P(rocess of Testing), R(euse of Software), I(ncident Management), E(nvironment), S(tandards), T(est Case Design Techniques).

Explain that the above information will not necessarily be presented within a single document, but may be spread across a set of documents, which could typically be labelled as:
- *corporate test strategy,*
- *specific site / location test strategies,*
- *programme test strategies (for a series of projects),*
- *project test strategies and*
- *also may be presented as part of the test plans.*

Describe why different strategies are appropriate for different application areas, giving examples from disparate areas such as safety-critical software and non-critical web-based software.

E.g. the software in an aeroplane guidance system is much more safety-critical than that in on-line shopping system. Or the security in an on-line shopping system is much more important than that in a car's navigation system (which has no contact with other computers). MARK

Explain that the results of any test process improvement initiatives are considered when creating the test strategy.

E.g. how can the testing be faster, cheaper or better? TP/306

3.1.3 Project Test Plan

- *Explain that, for this syllabus, a project test plan documents, for a particular project, the implementation of the overall test strategy.*
- *The project test plan will either confirm compliance, or explain non-compliance, with the test strategy.*
- *The project test plan shall normally be referenced from the project plan, which would identify the critical path.*

Explain that, in addition, a project test plan includes information to enable the tester:

- *To identify project testing **costs and timescales** to obtain approval for the release of funds and / or resources.*
- *To **i**dentify test cycles based on the software project release plan.*
- *To satisfy **m**anagement and users (customers) that adequate testing will be performed for this project.*
- *To define and communicate the contributions of **e**verybody who is expected to assist in delivering the testing for this project.*
- *To identify the project **d**eliverable(s) to be tested.*

MEDIC: M(anagement), E(verybody), D(eliverables), I(dentify), C(osts & Timescales).

3.1.4 Phase Test Plan

Explain that, for this syllabus, a phase test plan documents a detailed approach to a test phase (e.g. component test plan). The phase test plan describes in greater detail the implementation of the project test plan for a particular phase.

For instance, it would normally include a sequence of test activities, day-to-day plan of activities and associated milestones.

Phase Test Plan details

The phase test plan is the detailed document that will include all the information required to control testing for each phase.

The phase test plan will contain details such as:

- Details of modules/functions/processes to be tested in that phase
- Details of the business risks associated with those functions
- Test coverage requirements
- Test techniques to be applied
- Methods and tools
- Test specifications and execution requirements
- Details of the test environments to be used
- Testing schedule
- Details of any variations from the project test plan or the test strategy

CN/TMR-41

3.2 Test Plan Documentation

Explain how to develop Project Test Plans and Phase Test Plans in compliance with IEEE Std. 829-1998.

Test Plan Outline (IEEE 829):

1. Test Plan Identifier.
2. References. *Not officially part of IEEE 829.*
3. Introduction.
4. Test Items.
5. Software Risk Issues. *Not officially part of IEEE 829.*
6. Features to be Tested.
7. Features not to be Tested.
8. Approach.
9. Items Pass / Fail Criteria.
10. Suspension Criteria and Resumption Requirements.
11. Test Deliverables.
12. Remaining Test Tasks.
13. Environmental Needs.
14. Staff and Training Needs.
15. Responsibilities.
16. Schedule.
17. Planning Risks and Contingencies.
18. Approvals.
19. Glossary. *Not officially part of IEEE 829.*

CN/IEE 829 Test Plan Outline (Grey section 7, first folder)

3.3 Test Estimation

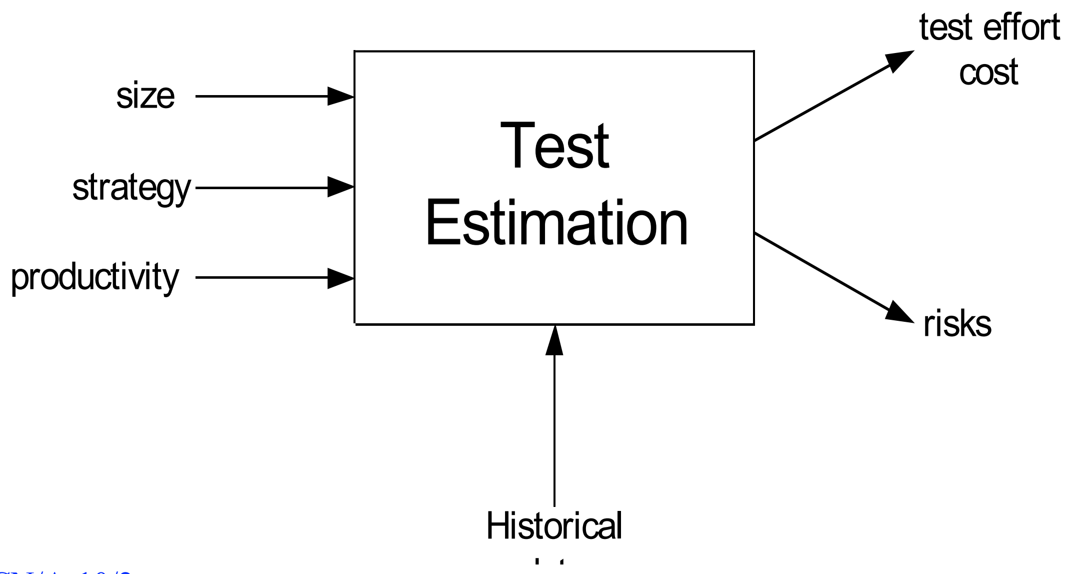
Explain that an estimate is an approximate calculation or judgement, typically based on the professional understanding of experienced practitioners.

Describe the need to estimate the number of tests, effort and / or time required for each phase, and number of iterations or cycles required for each phase, and any other costs, such as hardware, tools, etc. Explain that estimates include time for test planning and preparation as well as test execution. Explain why it is difficult to predict how long test execution will take when the quality of the software under test is not known.

Test process includes: plan, specification, execution, check & report and completion. This is worth memorising! PSERC (See "2.1 *Generic Test Process*".) TP/46

Explain how the duration of a testing activity can be estimated, to include:

- *Intuition, guesswork;*
- *Previous experience;*
- *Company estimating standards;*
- *A detailed work breakdown structure of all test activities;*
- *Formula based:*
  - *Function points;*
  - *Test points;*
  - *Related to software development effort (e.g. 40% for new software);*
  - *Metrics:*
    - *Estimate the number of iterations or cycles of test – debug – retest based on recent records of comparable test efforts;*
    - *Calculate the average effort required per test on a previous test effort and multiply by the number of tests estimated for this test effort.*



CN/A-10/2

Wide Band Delphi can be used for work estimates.
CN/A-04/1

Estimation related to software development:
- Testing takes 30-40% of development effort
- System/acceptance testing takes 25%

Estimation based on metrics:
- Number of test cycles based on recent records
- Time per test phase or activity in previous test projects

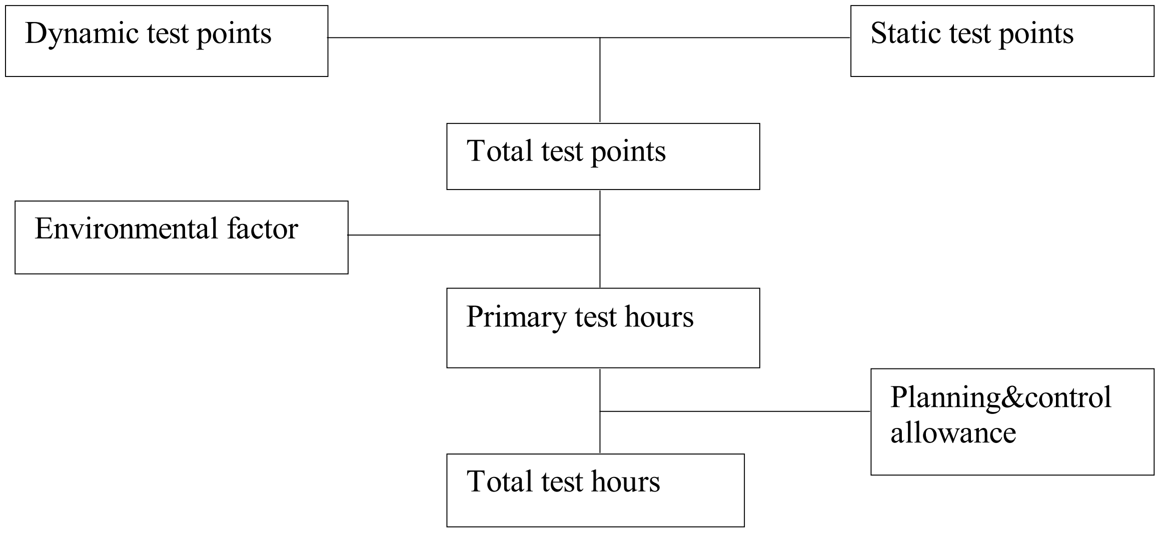
Estimation without function points:

X requirements critical	* A * 1,25 = …
Y requirements normal	* A * 1,00 = …
Z requirements low	* A * 0.75 = …

…with A = average time to test a requirement
Apply environmental and productivity factor, plus planning and control allowance → testing time.

Estimation with function points. Test Point Analysis (TPA). Technique to estimate a structured system and/or acceptance test

Needed:
1. Size of test object: # function points, corrective factors:
   a) complexity
   b) interfacing
   c) uniformity
2. Importance of subsystems/functions
   a) user importance
   b) usage intensity
3. Quality attributes:
   a) usability
   b) efficiency,
   c) risk level
   d) …etc
4. Productivity:
   a) Environmental factors: test basis, development test, tools, environment, …
   b) Productivity figures per organization, skills test team, historical data



Estimation by work breakdowns and Wide Band Delphi:
- From test strategy and (standard) test approach determine all deliverables to be developed
- A WBD (Work Break Down) contains the tasks needed to produce the deliverables (test plan, design, script, reports) and other tasks (reviews, test environment, management and control)

Wide Band Delphi
- Input: WBDs, data from previous projects
- Moderator
- Agreement on % maximum deviation
- Explain the WBS and discuss requirement quality, complexity, etc.
- Each participant estimates (each task max X hours) and documents assumptions (supported by an “influencing factors” checklist)
- Moderator calculates averages and deviations and collects assumptions, then re-distributes
- Discuss outcome (based on criterion % max), the low and high explain their reasoning
- If necessary, a new estimation round is carried out

Test execution estimation: include estimation for rework!

Attention points:
- Include a contingency budget? 20%?
- What about estimation for tools and test environment?

Do not give in easily when the schedule is under pressure:
1. Relax entry criteria, but understand and communicate the risks involved
2. Add staff (but do they have the right skills?)
3. Reduce test execution time, accept lesser quality (relax exit criteria) → risk based, or overall lower coverage
4. Reduce the number of features in the product

MARIO

3.4 Scheduling of Test Planning

Present reasons why planning should be done as early as possible, e.g. to give adequate warning to others who will be involved and to give early visibility of potential problems.

Describe the benefits of using test planning to assist in the preparation of the development delivery plan e.g. defining the priority given to a test activity could be used to determine which software components are developed and delivered first.

Explain the advantage of releasing test plans in stages (iterations) if not all information is available in time, in order to avoid delay.

Faults in test plans can therefore be found earlier.
Scheduling benefits:

- problem areas are now visible
- early test planning and scheduling will assist in development delivery plan, e.g. most important functionalities delivered up-front
- dependencies are made clear- impact of slippage is more easily identified
- structured progress tracking (management) becomes possible

CN/A-10/11

3.5 Test Progress Monitoring and Control

Explain different means of monitoring the progress of the testing, to include the monitoring of incidents and test cases and the use of statistical methods.

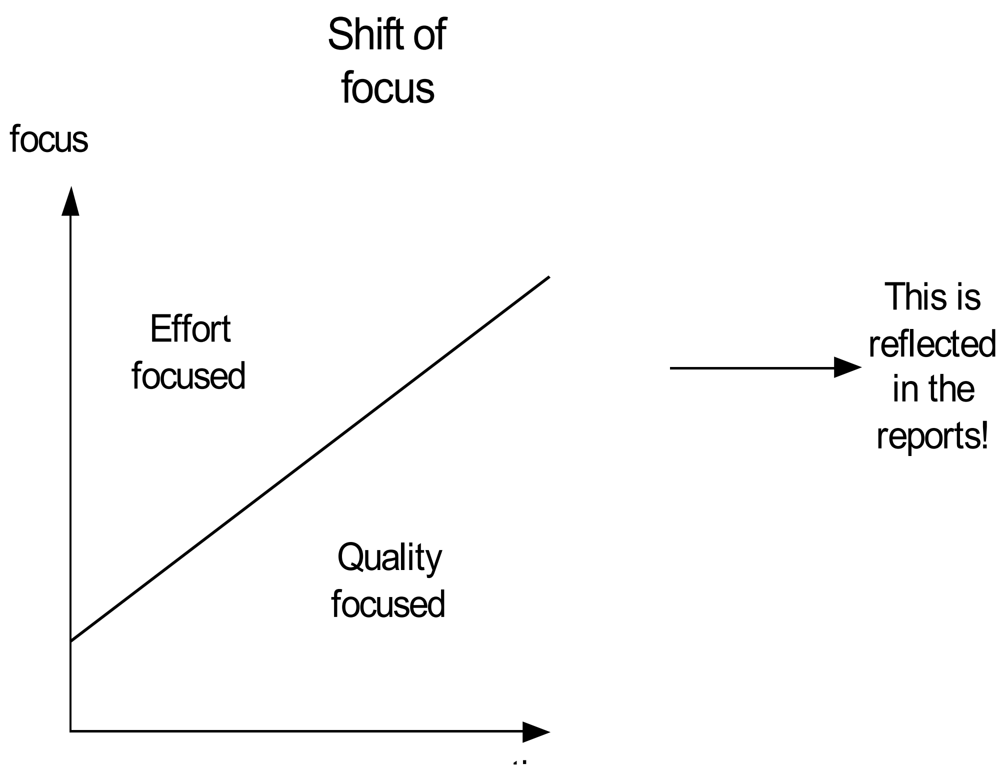
For example, test results can be summarised for management as faults found per week. This can be show on a bar chart to indicate progress (towards the end of the test phase, the number of reported faults should decrease, assuming developers are continually fixing the software and providing the testers with the most recent stable version). MARK

Test execution monitoring

- Number of test cases executed as a percentage of all the test cases
- Number of high priority test cases executed as a percentage of all the high priority test cases
- Use of statistical methods

Regular meetings should be scheduled in order to track progress against plan.
Any milestones that are missed, or are in danger of being missed, need to be addressed.

CN/TMR/81-82



CN/A-12/2

Explain that test reports are used to communicate test progress and that these reports may be tailored to the different recipients. Describe how test progress can be presented graphically and using numerical tables.

Identify the requirement to control test activities to minimise divergence from the test plan and describe methods of control, to include reviewing the priority of tests, obtaining additional resources, extending the release date and (only with project management support) changing the test completion criterion.

How to deal with slippage:

- discuss with all stakeholders
- defer release date (if not fixed)
- allocate additional resources
- execute highest priority tests in the time available
- revisit the test completion criteria
- or…deliver less functionality

CN/A-12/5

4 Testing and Risk

4.1 Introduction to Testing and Risk

Define risk as the chance that a problem may occur in the future. Explain that risk is a combination of both the likelihood of a problem occurring and the impact of the problem. Such problems may impact the product (e.g. a software failure that takes the delivered system out of operation) or the project (e.g. extend the delivery timescales).

Describe typical risks (both product and project) to be considered such as those related to safety, economic, security and political and technical factors and give examples of each of these.

Example risks:

Category	Annoying	Hindering	Damaging	Catastrophic
Safety: aircraft.	Dashboard light flashes continuously.	Need to restart computer every 36 hours.	Error in guidance leads to extra wear on wings.	Undercarriage does not always respond to software signal.
Economic: market analysis tool.	Companies only identified by codes in UI.	Not possible to delete old companies.	All profit figures always rounded up.	Profit displayed for incorrect company.
Security: airport baggage scanner.	Software takes too long to start (once per day).	Tuning too high: alerts to padlocks on baggage.	Scan misses an acceptable amount of metal (e.g. razor blades).	Scan can damage organic material close by.
Political – trial release to new customer.	User interface contains spelling mistake.	User interface requires too many “clicks”.	Application crashes once a day with loss of up to 30 minutes work.	Application corrupts database.
Technical – printer software.	Blank page used at start of print run.	Cancelling print run results in many pages of “junk” characters.	Arial font rendered as Wingdings.	More than one page of graphics requires printer reboot.

MARK

Explain that risks can be treated in either a qualitative or quantitative manner.

Qualitative: relating to quality. Involves categorising risk (high / medium / low).
Quantitative: relating to quantity. A mathematical way of determining risk and impact.

For detailed explanation, see 4.2.3 “Risk Analysis”.

MARK

Explain that risk analysis can be used to:

- *Target testing – different risks can be addressed by different types (and depths) of testing.*
- *Prioritise testing – give the testing of higher risk areas higher priority.*
- *Describe the risks of delivering a system – if testing is cut short (or not done at all) then the risks that have not been addressed by testing will remain.*

Explain that testing can be used to:

- *Mitigate (reduce) risks – a primary way of mitigating product risks is to identify faults in a product (testing) that are subsequently removed (debugging). Project risks may also be mitigated by the appropriate choice of test strategy.*
- *Inform the risk assessment – the results of testing, such as high/low fault rates in certain parts of a product can be used to improve the accuracy of the risk analysis.*

4.2 Risk Management

4.2.1 Introduction to Risk Management

Introduce the core activities of risk management as comprising risk identification, risk analysis and risk mitigation.

- Risk Identification: looks at ways of establishing what the risks are and where they are.
- Risk Analysis: looks into the critical, complex and potentially error prone areas. Identifies the likelihood and impact. CN/A-06/2
- Risk Mitigation: tests built to try to reduce likelihood of risk. MARK

For more details, see sections below.

Explain that ideally all stakeholders should be directly involved at all stages of risk management.

Who owns the risks?

- All key stakeholders should be involved at all stages:
  - Who is responsible?
  - Who has a problem when things go wrong?
  - Who needs the system at their work?
- Note that testers (and requirements engineers) have a responsibility to identify the risks.
- Ultimate decision is with the business owner of the system.
  - The can only make this decision if they have objective information – that is the tester’s role.

CN/A-06/2

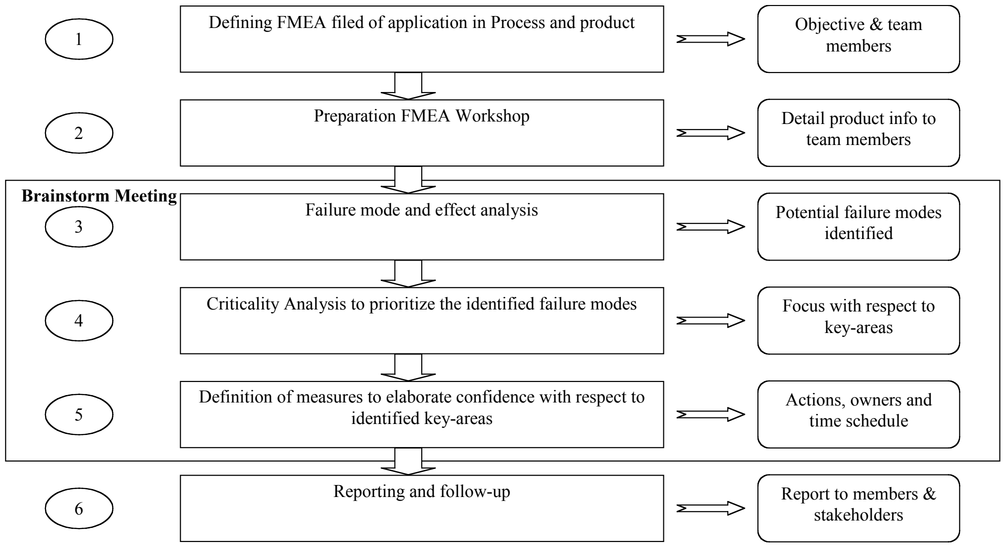
4.2.2 Risk Identification

Introduce the following techniques that are used to identify risks:

- ***e**xpert **i**nterviews;*
- ***i**ndependent **a**ssessment;*
- *risk **t**emplates;*
- ***l**essons learned;*
- *risk **w**orkshops;*
- ***b**rainstorming and*
- ***c**hecklists.*

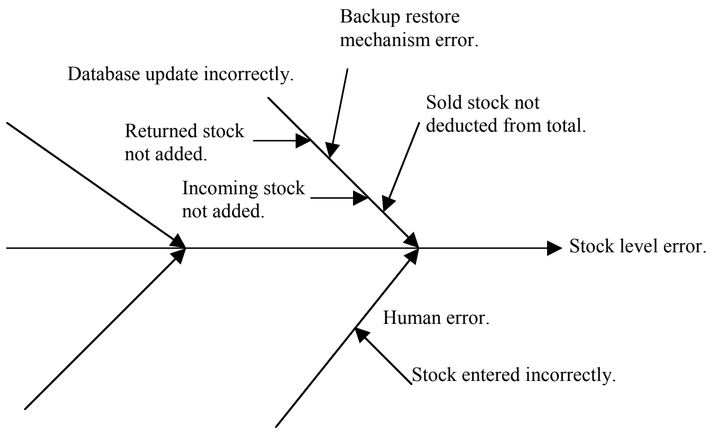
I BITE CLAW or **ITALIC WEB**.

Can also use FMEA (Failure Mode and Effect Analysis). CN/A-06/3-4.



1
Defining FMEA filed of application in Process and product
Objective & team members
2
Preparation FMEA Workshop
Detail product info to team members
Brainstorm Meeting
3
Failure mode and effect analysis
Potential failure modes identified
4
Criticality Analysis to prioritize the identified failure modes
Focus with respect to key-areas
5
Definition of measures to elaborate confidence with respect to identified key-areas
Actions, owners and time schedule
6
Reporting and follow-up
Report to members & stakeholders

FEMA Process.
CN/A-06/3



Backup restore mechanism error.
Database update incorrectly.
Sold stock not deducted from total.
Returned stock not added.
Incoming stock not added.
Stock level error.
Human error.
Stock entered incorrectly.

Ishikawa / Fishbone diagram. Each branch is made up of sub-branches which break down the risk further.

Give examples of the appropriate use of the above techniques.

- Expert interviews: interviewing end users to identify cost (damage, etc.) if specific parts fail.
- Independent assessment: ask an expert in the field, e.g. ask an avionics expert where most faults occur.
- Risk templates: includes factors such as likelihood and impact.
- Lessons learned: review where faults where found in previous project, a root cause analysis => prevention.
- Risk workshops: possibly straight after a requirements specification walkthrough meeting.
- Brainstorming: can use fishbone diagram.
- Checklists: list of questions to answer, e.g. what would be the impact on the business if this module failed?

MARK

4.2.3 Risk Analysis

Introduce risk analysis as the study of identified risks.

After risks are found, they need to be analysed. MARK

*Explain that risk can be calculated as (frequency * severity), and that if the frequency and severity (likelihood and impact) can be described quantitatively then a quantitative value for risk (the exposure) can be calculated. Point out that it is most often the case that frequency and severity will not both be quantitatively defined and so direct calculation of risk is not possible. In these situations risk is defined as one of a number of classes or categories.*

- Value of a certain risk can be calculated as frequency * severity.
- Severity is a combination of likelihood and impact (quantitative).
- Both frequency and severity values are unlikely to be available.
- Risks are then defined by class (type) or category (qualitative).
- Usually this is based on perceived risk.

CN/A-06/5

<u>FMEA Criticality analysis</u>

- Assign Risk Priority Number (RPN) to identified risks
- RPN:
  - Probability x Seriousness x Detecting difficulty
  - Each on a 1-10 scale (agreed calculation criteria)
  - 0<RPN<1000, guidance
    - RPN<50 :no action
    - RPN<150 :action desirable
    - RPN>150 :action a must
    - RPN>500 :critical

CN/A-06/6

Often the frequency and severity of a risk are not quantitatively defined (e.g. frequency 56%, severity €123 000 / day) so they are categorised, e.g. High / Medium / Low. MARK

Emphasise that unless trusted risk metrics are available, the analysis will be based on <u>perceived</u> probability and consequence. Explain that because risk analysis is usually based on personal perceptions, it is common for views to differ. Compare the different viewpoints of a software project manager, a developer, an end-user and a tester. Explain that the results of the risk analysis should always include the level of uncertainty in the resultant risks.

<u>Perceived</u> likelihood and impact:

- Views differ.
- As a consequence there is a level of uncertainty.
- This will diminish as the project progresses.

CN/A-07/2

<u>Viewpoints:</u>

- Software project manager wants the software to be delivered on time.
- Developer often wants to try out new things or do a "beautiful" design.
- End-user is interested in Return On Investment.
- Tester is focussed on quality.

MARK

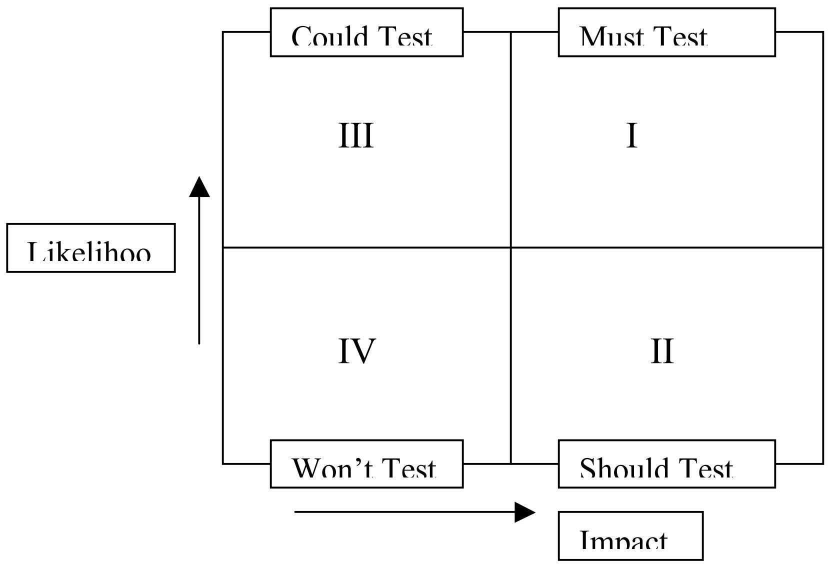
Risk analyses, when applied to software products, generally need to be broken down into lower level risk categories. The most common categories would relate to:

- *Functional aspects (particularly important functions or processes to be supported).*
- *Non-functional aspects, e.g. performance, security, usability, etc.*

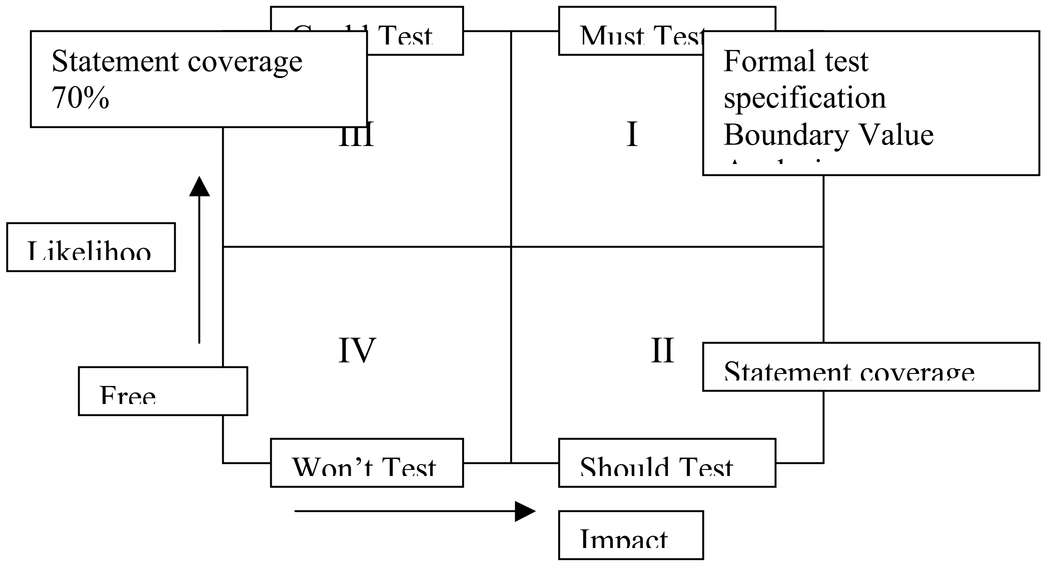
Explain that a variety of approaches to risk analysis are possible, which employ varying degrees of rigour, from the calculation of a quantitative value to the simple declaration of class of risk by the customer.

For example, a functional risk matrix (MoSCoW priorities), Rex Black Classification Scheme (where each risk is given a weighting for impact, exposure and likelihood) or stakeholders can be asked to scale each risk from 0 – 9.

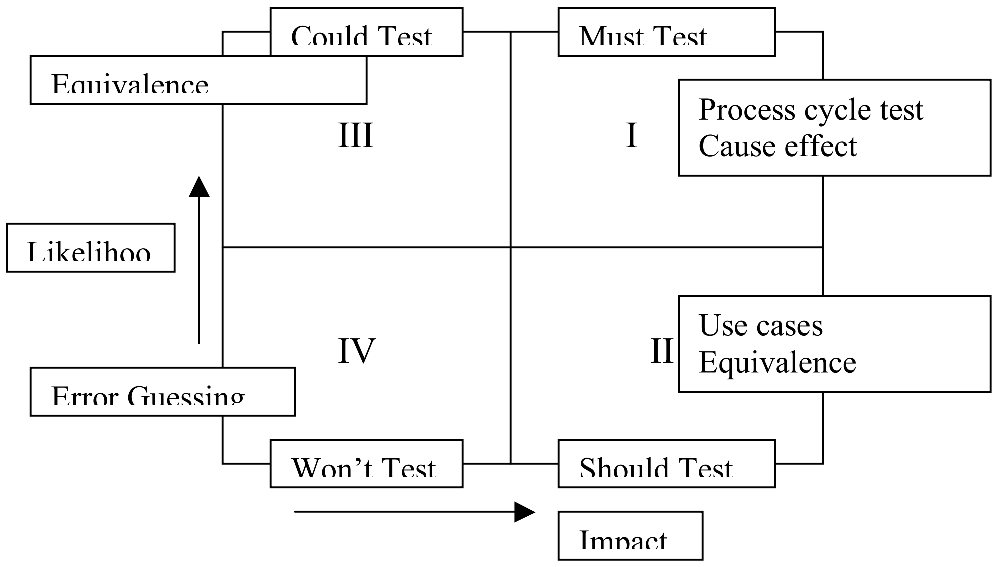
<u>Functional risk Matrix</u>
MoSCoW (<u>**M**</u>ust test, <u>**S**</u>hould test, <u>**C**</u>ould test, <u>**W**</u>on't test)



Example Low level test



Example high level test



CN/A-06/6, CN/A-07/2-3

Give examples of both qualitative and a quantitative risk assessments.

Qualitative: risk likelihood and impact are categorised high / medium and low.
Quantitative: risk likelihood is expressed as a percentage and impact is expressed financially.
MARK

<u>Risk management summary</u>

- Functional risks
  - Factors score sheet
  - Risk matrix
- Non-functional risks
  - Quality characteristics
  - Structured questionnaire

CN/A-07/8

Explain that risk analysis should be ongoing throughout a project and that the results from testing can be used to inform the risk analysis, such as allowing risk values to be increased/decreased and uncertainty levels to be reassessed. For example, where a tested component is found to contain considerably more than the expected number of faults, it may be decided that the component's quality is less than expected, giving rise to an increase in the likelihood of the component failing, and a subsequent increase in the risks associated with this component. In response, it may then be decided to increase the testing necessary for this component to mitigate the increased risks (see below).

4.2.4 Risk Mitigation

Explain that risk mitigation is the activity concerned with the response to the analysed risks.

Describe the possible responses as

- *doing nothing,*
- *sharing risk,*
- *taking preventative action to avoid or reduce risk and*
- *planning for contingent action.*

Possible responses:

1. Doing nothing
2. Sharing the risk (e.g. insurance)
3. preventive action (e.g. change architecture or change development process)
4. detective action (test)
5. corrective action (planning for contingency action)

CN/A-06/7

Explain that the choice of response will depend on the benefit of removing or reducing a particular risk. Explain that in this respect testing is a form of preventative action that reduces risk by identifying faults that are subsequently removed.

Cost / benefit ratio: cost of failure in operation vs. cost of finding and fixing now. (Also need to consider what the possibility of new faults being introduced is.) MARK

Explain that knowledge of risks can be used to determine the content of the test strategy. Explain that this knowledge can be used in two ways:

- *First, the level of risk can be used to determine the 'level' of testing to be performed. This approach is used by most safety-related standards, where specific test case design techniques and test completion criteria are mandated for each level of risk, with four levels typically being used. For instance, all components deemed to be at the highest level of risk may have to be tested to achieve 100% branch coverage.*
- *Secondly, the type of risk can be used to determine the 'type' of testing to be performed, as certain types of risks are known to be amenable to being mitigated by certain types of testing. For instance, a risk associated with the product's user interface may be mitigated by performing more extensive usability testing.*

This also helps to identify the priority of each of the tests, which is useful when budgets are cut.

<u>Risk mitigation summary</u>

- Select testing techniques
- Defining test levels (responsibility)
- Content V-model, completion criteria

CN/A-07/7-8

Give examples of how project and product risks can be addressed by decisions documented in the test strategy based on both the level and type of risk, such as which test phases to include and which test case design techniques and test completion criteria to use.

Test project risks should contain:

- probability (low, medium, high or numeric)
- impact type (lead time, effort, quality)
- impact size (low, medium, high or numeric)
- trigger (how will we know "it is happening")
- contingency plan

CN/A-09/2

Test case design technique = a method used to derive or select test cases

These to ensure the activities are:

- measurable
- controllable
- repeatable

Test techniques provide an understanding of the complexities imposed by most systems. The use of techniques forces testers into thinking about what they test and why they are testing.

Static vs. Dynamic: dynamic test case *design* is carried out at the same time as the static *review* activities. Early in the V-model lifecycle. The dynamic test cases are executed once the system has been built and delivered into the test environment.
Static testing are primarily techniques aimed at preventing faults being propagated to the next phase or into the test environment. Dynamic techniques (black or white) are intended to find faults in the translation of the specifications into the actual system.

Systematic vs. Non-systematic:
Systematic techniques are based around a set of rules that make the activity mechanistic, repeatable and measurable.
Non-systematic techniques are based on the experience of the tester (e.g. Exploratory testing). The biggest draw back with these techniques is the lack of documentary evidence and therefore repeatability of the test scenarios.

Functional vs. Non-functional:
Functional test techniques relate to establishing what the system does. Do the functions work?
Non-functional techniques focus on establishing how the system does what it does, i.e. is it secure? How quickly does it respond? Is it reliable? ..Non-functional techniques often use tools to execute the tasks with supporting tools analyzing the dynamic system activity, e.g. memory use, concurrent threads, …

Advantages/Disadvantages of test techniques:

Advantages	Disadvantages
Objectivity	Requires training to some degree
Formal coverage measures	Time to implement – culture change
Early defect finding	Everyone must be “bought in”
Traceability	Not seen as useful for all applications
Coverage independent of the tester	Takes more time than less formal design
Way to differentiate test depth based on risks using different test techniques	Does not cover all situations (error guessing still useful)
High level of re-use (re-usable testware)	Little use of domain and product knowledge of tester
Repeatability and reproducibility	
Audit trails	
Higher defect finding capability	

MARIO

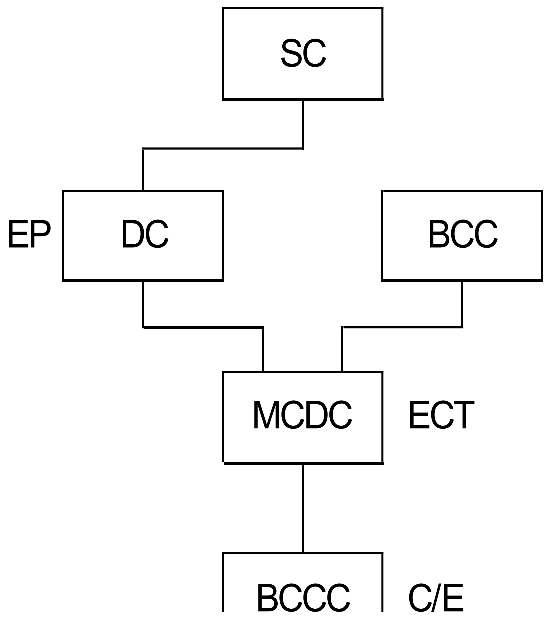
Explain that where testing is to be used to mitigate risk, and where there is a limited budget for testing, then it will normally be necessary to prioritise the risks so that the higher risks are addressed by testing first. This may mean that lower priority risks are not addressed by testing if the testing budget or project timescales do not permit it. In this case project management will know that the unaddressed risks remain and can take this into account when deciding whether to release the product and so accept the remaining risks, or not.

This implies that completion criteria are very important. Otherwise, how do you know that it is “safe” to stop? MARK

5 Test Techniques

5.1 Functional / Structural Testing Techniques

Provide an introduction to the test case design techniques defined in BS 7925-2:1998, classification tree method [see M. Grochtmann and K. Grimm, Classification Trees for Partition Testing, Software Testing, Verification and Reliability, 3:63-82, 1993] and to path testing [see Beizer's Software Testing Techniques]. Provide practical experience of applying Equivalence Partitioning, Classification Tree Method, Boundary Value Analysis, State Transition Testing (to include Chow's coverage measures), Statement Testing and Branch Testing to create tests. Introduce an approach to requirements-based testing. Explain that structural test case design techniques are nearly always linked with a corresponding structural test completion criterion.



BCC	
BCCC	
C/E	
DC	
ECT	
EP	Equivalence Partitioning
MCDC	
SC	

BRYAN

See also Appendix D: Summary Of Test Techniques.

5.2 Non-Functional Testing Techniques

Explain the techniques used for reliability testing, usability testing [details available at: http://www.testingstandards.co.uk], volume, performance and stress testing [see Myer's The Art of Software Testing for descriptions of the last three techniques].

SUMI (software Usability Measurement Inventory)

- Well founded questionnaire (50 questions) : based on practical research.
- Referred to in ISO 9126 and ISO 9241.
- Quantitative, objective information of users' subjective attitude to six usability aspects.
- Supported by an analysis tool.

SUMI - Metrics

- Affect – user's feeling about interacting with the product.
- Efficiency – user's perception of efficiency of task performance.
- Helpfulness – view of how communicative product is.
- Control – user's feeling about product response in a normal manner.
- Learnability – How quick does the user becomes familiar with product.

Cost/benefit SUMI

- Easy to use; Fast well founded results
- Limited detailed analysis possible
- Objective indicator
- Low cost (3days, including reporting)
- Late lifecycle
- Minimum of 10 users doing the "same' tasks.

CN/C-16

Use Cases

A user driven development and testing approach.

- Identification and specification of user scenario's (High level use cases)
- User-centred approach
- Use case: sequence of transactions in a dialogue between user and system with a specified result.
- A use case depicts a typical way of using the system and nothing more.
- A use case has a beginning, a main body and an ending (=expected result)
- A use case fits on 1-2 pages.

Cost / Benefits

- Main tasks/functions are being tested (disadvantage: limited coverage)
- Preferably splitting test specification and execution (ask user to run the test case)
- Preferably early involvement (static testing)
- User focussed: real-world defects
- A different view and thus other defects.

CN/C-17/1-4

How to make Use Cases:

1. Identification of users (actors) of the system + priority
2. Identification of use cases for each actor + priority
3. Identification of paths for each use case
   (basic scenario, deviation scenario, failure scenario)
4. Decide on use cases to elaborate
5. Develop test procedure

CN/C-17/2-3

Note: Use cases are about validation: are we making the right product?
MARK / BRYAN

Usability (lab) testing

Testing with real users.

- Identifies usability problems that are very likely to occur in real use.
- End users performs tasks with a product, while being observed by usability specialists.
- User are interviewed after the test (SUMI?)
- In short: real user experience.

Cost / Benefits

- Representative for real use
- Time consuming
- Knowledge: usability expertise is needed to interpret results
- Users: 3-10
- Costs: "Expensive" with lab, high benefits
- Life cycle phase: working prototypes, working product.

CN/C-17/5-8

Maintainability:
A set of attributes that bear on the effort needed to make specified modification:
- Analysability
- Changeability
- Stability
- Testability
Reference: CN/C-03/1

Types of maintenance:
perfecting (enhancements), preventive, adaptive (changes in the environment), corrective + not planned/ad-hoc corrective

CN/C-03/5

Heuristic evaluation

Usability: narrow focus: very local, e.g. navigation/help in screen
broad focus: can users carry out their tasks → use cases
→ effectiveness, efficiency and attractiveness

CN/C-04/1 and TP/269

Heuristics:

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Design useful error messages
6. Recognition rather than recall
7. Flexibility and efficiency in use
8. Aesthetic and minimalist design
9. Error prevention
10. Help and documentation

CN/C-04/3

Component testing:

Different Coverage's:

- Statement coverage
- Decision coverage ≈ Branch coverage
  Based on decision points — branches linked to decision points + entry and exit points
- Branch Condition Coverage (each single condition)
- Branch Condition Combination Coverage (all possible combinations) → C/E graphing
- Modified Condition Decision Coverage (every condition determines twice the result)
  (min #testcases for 100% MCDC = #decision +1)
- Path Testing → lot's of test cases
  Also: test business paths through a system

CN/C-05/2-5

Integration Testing in the Large:

About: interoperability, fault-tolerance (on external interfaces)

BRYAN

C/E graphing:

Steps:

1. determine causes (single conditions, positive way (don't use NOT), ranking)
2. determine effects
3. establish decision table → possible simplification
4. determine test cases (incl. expected results)

CN/C-08/1-2

Reliability:
Capability of the software product to maintain a specified level of performance when used under specified conditions:

- Maturity (MTBF, #breakdowns)
- Fault-tolerance (incorrect input) at system level
- Recoverability (MTTR, mean restart time)

CN/C-09/1

Approach: Fault avoidance
Fault removal
Fault tolerance

Static techniques (for reliability): reliability assessment, metrics (static code analysis, ISO9126 part 2/3) design & code inspection
CN/C-09/2

Dynamic techniques (for reliability): syntax testing, random testing (operation profiles), volume, load & stress testing (system boundaries)
→ Software Reliability Engineering Testing
→ achieving reliability
→ reliability evaluation
CN/C-09/4-5

State transition testing:

The cases should include:	starting state
	event
	expected action
	expected next state

CN/C-11/1

0-switch: transition of a component from one state to another
1-switch: transition of a component through two states
CN/C-11/2

Performance testing (efficiency according to ISO9126):
Capability of the SW product to provide appropriate performance, relative to the resources used, under specified conditions:
- Time behaviour (response time, processing time, throughput rates)
- Resource utilisation (appropriate type and amount of resources)

→ higher level (system testing)
CN/C-12/1

Stress testing: subjecting the program to heavy load or stress. Heavy stress is a peak volume of data processing encountered over a short span of time.
Volume testing: Subjecting the program to heavy volumes of data → beyond operation profiles!
CN/C-12/2

Process Cycle Test: → suitability
Validation, used in acceptance test
→ test suitability with respect to organizational procedures, e.g. task descriptions, user manual → possible usability problems.
Needed: process description or flows
CN/C-18/1 + BRYAN

In depth PCT: measure 2 (path combinations, including 1 decision point)
Normal PCT: measure 1
CN/C-Handouts/36

Summary of typical test techniques:
Component test: BVA, C/E, ECT, (STT), (Syntax), Structural testing
Integration small: EP, Semantic, STT, Syntax (I/F), Use Cases
System Test: EP, C/E, CTM, ECT, Semantic, STT, Syntax (UI), Error Guessing, ET
Integration Large:
Acceptance: PCT, Semantic, Use Cases, EP, ET, Error Guessing
TP/222-249

Maintainability: Component
Performance: System

Usability:	System →	heuristic evaluation
	Acceptance →	use cases
		usability lab
		SUMI

Reliability: System
BRYAN

5.3 Dynamic Analysis

Explain that dynamic analysis provides run-time information on the state of executing software, normally achieved by instrumenting the program under test. Commonly used to monitor the allocation, use and de-allocation of memory, flag memory leaks, unassigned points, pointer arithmetic and other errors difficult to find 'statically'.

5.4 Static Analysis

Explain that static analysis involves no dynamic execution of the software under test and can detect possible faults such as unreachable code, undeclared variables, parameter type mismatches, uncalled functions and procedures and possible array boundary violations.

Explain that any faults found by compilers are found by static analysis. Compilers find faults in syntax. Many compilers also provide information on variable use, which is useful during maintenance.

Describe in detail the concepts on which data flow analysis is based. Explain that data flow analysis considers the user of data on paths through the code, looking for possible anomalies, such as 'definitions' with no intervening 'use' and the 'use' of a variable after it is 'killed'.

Explain the use of, and provide an example of, the production of a control flow graph for a program. Use the control flow graph notation as described in Hetzel's The Complete Guide to Software Testing.

Introduce the use of complexity metrics, explaining the calculation of lines of code (LOC) and cyclomatic complexity.

Can also calculated nested levels (e.g. maximum nesting).
Cyclomatic Complexity: Number of decisions + 1. MARK

Process: coding → static analysis → compiler → dynamic testing
CN/C-02/3

Checks the following:
- coding standards
- style guide
- language validation (syntax, constraints and semantics)
- metrics: function based, file based

CN/C-02/4+7

Static analysis =automated technique to "walk through" the source code and detect constructs non-complying with pre-defined rules. Eg: Coding standard, Style guide (compiler is also a static analysis tool)

In time, Static Analysis is done after coding and before compilation – often Static Analysis is done after compilation though.

Reviews are also part of static analysis

60% of the SW faults that were found in released SW products could have been detected by means of STATIC ANALYSIS (including reviews)

Syntax is the formal set of rules by which valid language statements are constructed from the basic tokens of the language.
Semantics are the rules governing how the language statements are interpreted
Constraint is a syntactic and semantic restriction on the rules of construction

Why static analysis:
1. Programmer makes mistakes
2. Programmer misunderstands the language
3. Compiler does not react as expected
4. Compiler contains errors

5. Program does not react as expected (run-time errors)

There are five levels of Static Analysis; it is best to start with level 1 as a start-point:
1. Adhere to programming best practices
2. Can easily be used and maintained by others
3. Can be integrated with other SW parts and made operational with little debugging and rework
4. Offers features and functionality that is expected and required by the SW user
5. Leads to a software product that is reliable and does not fail operation during use

Function based measurement (Based on the content of a single function):
- Number of function calls
- Cyclomatic complexity

File based measurement (based on the content of a file):
- Number of variables declared extern
- Number of variables declares as static
- Number of functions in the file

MARIO

5.5 Non-Systematic Testing Techniques

Make students aware of the benefits of including non-systematic testing techniques, such as exploratory testing, ad hoc testing and error guessing, in their test process.

Exploratory Testing

Team based approach, formal test process, skills required.
- Test charters (objectives)
- Product exploration
- Test design
- Test execution
- Reviewable results (notes)
- Heuristics

Test Charters
- What (scope)
- What not (not within scope)
- Why (questions to be answered)
- How (brainstorm)
- Expected problems
- Reference (diagrams/modes)

Exploratory Testing is simultaneous exploration, design and execution.
CN/C-14/3

<u>Error guessing</u>
Should be carried out after more formal coverage has been achieved.

- No plan, no documentation
- Random, trying out
- Any fool can stumble across bugs…
- Domain/product expertise
- Hot spots/high risk areas (after more formal techniques)
- Low risk areas (as only technique)
- Sample to drive further testing
- Complementary to more formal techniques
- Friday afternoon

<u>Error guessing – Disadvantages</u>

- No clear objectives
- Coverage unknown
- Defect reproducibility/repeatability
- No re-usable test designs (testware)
- Finding defects is down to "luck"
- Hard to manage

<u>Exploratory Testing vs. scripted testing</u>

```
Fully          Exploratory testing                          Ad-hoc
Scripted   <---------------------------------------------->  testing
Testing                                  Error guessing
```

CN/C-14

5.6 Choosing Test Techniques

Explain how the choice of which test techniques to use is made. Explain that it may be made based on regulatory standards, experience, and/or customer/contractual requirement. Describe typical standards that mandate particular techniques, to include as IEC/ISO 61508, DO-178B, Railway Signalling standards, Nuclear industry standards, Pharmaceutical standards and MISRA guidelines for motor vehicle software. Introduce the subsumes ordering of techniques. Describe the current state of knowledge with regard to the effectiveness of different test techniques.

Choice can be based on, e.g.:

- risk
- documentation available?
- knowledge of testers
- time and budget
- supporting tools
- previous experience
- standards to be complied

- customer/contractual requirements

CN/A-08/2

6 Reviews

6.1 Introduction to Reviews

Present the arguments for using reviews, to include fault-finding and cost effectiveness.

Main objectives of review:

- find <u>defects</u>
- find defects <u>earlier</u>
- remove <u>causes</u> from the process

CN/B-01/1

There are three types of formal reviews:

<u>Inspection</u> – formal individual and group checking using sources and standards, according to detailed and specific rules (checklists), support the author, find defects → **verification**

<u>Technical review (= peer review)</u> – group discusses a document and tries to reach agreement about the technical content (approach), how things will be done.

<u>Walkthrough</u> – author guides the group through a document and his thought processes, so all understand the same thing and reach consensus on changes to make (often with people outside the software project on higher levels) → **validation**

All three formal review types have a different focus, however they aim to explain and evaluate the contents of the document in order to improve its quality. In other words, they aim to find defects. The types of defects found differ for the three review types.

In addition to the previous formal reviews, the IEEE 1028 standard mentions two other types of **formal** reviews:

<u>Management review</u> – a project review in which the project status is evaluated against the plan (resources, cost, progress, quality) and appropriate actions are defined.

<u>Audit</u> – independent evaluation of project and product using defined criteria (e.g. internal audit by QA-group)

There are also informal reviews:

<u>Informal review</u> – a type of document review that doesn't follow a formal process and entry/exit criteria. Ask a colleague to provide comments. Rarely involves a meeting.
MARIO

Review advantages:

- Improves product quality (fault-finding).
- Improves cost-effectiveness (faults are cheaper to fix earlier).
- Improves quality attitude in team – people are more focused on quality.
- Trains people on how to produce good documentation, because everyone can learn from the mistakes of others.

MARK

Explain that any written product can be reviewed. This could include all code and written documentation e.g. requirement specifications, design documents, code specifications, test plans and all test documentation.

Explain that for each review technique the primary objective is to find faults.
This is why I added this to each of the reviews below as the primary goal.

Introduce IEE Std. 1028-1997 Standard for Software Reviews.
This document introduces three of types of review covered by the Practitioner course (not Informal), plus Management Reviews and Audits.
IEE Standard for Software Reviews, 1028-1997 (Grey section 7, first folder)

6.2 The Principles of Reviews

Explain that reviews are ideally performed when the source documents (those documents that specify the requirements of the product to be reviewed) and those standards to which the product must conform are available. Without source documents, reviews are generally limited to finding faults of consistency and completeness within the document under review.
So, without external documents, the reviews can only focus on the document itself without any "points of reference". MARK

Describe the three possible outcomes of a review:

- *First, it may be decided that the product can be used as it is.*
- *Second, it may be decided that there are issues that need to be dealt with, but it is unnecessary to perform a further review on the product.*
- *Finally, if the fault level is found to be too high, then a further review is considered necessary on the product after the issues have been dealt with.*

Decision of the review can be:

- Document is OK, as is,
- Document is OK, after moderator has checked,
- Document is OK, after all participants have checked,
- New inspection required.

CN/B-04/3

Explain that those involved with reviews often have specific roles. These may be:

- *The author of the product,*
- *The scribe, who records the issues raised at the review meeting,*
- *The chair, who runs the meeting and*
- *The presenter, who leads the meeting through the product.*

The chair may also be referred to as the moderator. MARK

A trainee may also present to learn the process. CN/B-02/1

There will often be other reviewers, who simply review the product. There may be a leader for the complete review process. Individual reviewers may be required to look for specific faults and, if so, are often given checklists of fault types to look for.

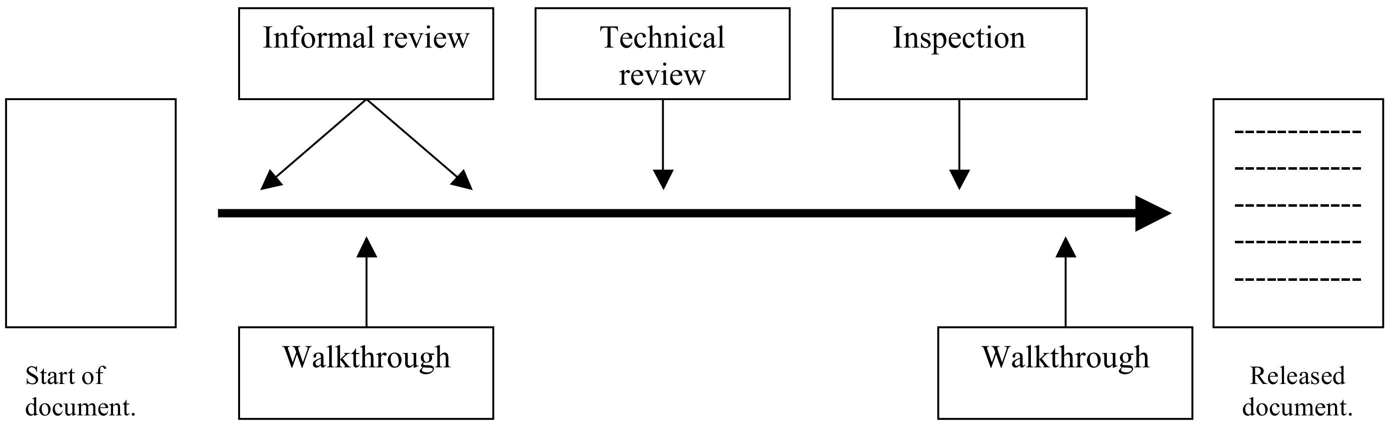
These are called roles, e.g.…
- Compliance to higher level document.
- Compliance to procedures and standards.
- Compliance to related documents.
- Testability.
- Completeness.
- Others…

CN/B-02/3 & MARK

The following roles can be identified:
- author
- moderator
- reviewer
- scribe
- presenter
- trainee

CN/B-02/1

Explain that more than one of the review techniques may be employed on a single product. For instance, you could start with an Informal Review, make changes as a result of the review, and then perform an Inspection.



TP/137

Explain that when reviews (a form of static testing) are used on code then they should normally be followed by dynamic testing, which can find those types of fault that cannot be found by static testing.

E.g. memory leaks. MARK

Reviews and dynamic testing will find different kinds of faults.
BRYAN

Explain that there are different types of review and that the different types have varying levels of formality and some have secondary objectives. The different types should be compared, their relative strengths and weaknesses highlighted and situations where each may be applicable identified.

Moderators always need to check that the focus of the review is on the document, not the author.

	Informal	**Walkthrough**	**Technical**	**Inspection**
Formality	Informal.	Rather formal.	Less formal.	Very formal.
Primary Goal	Find faults.	Find faults.	Find faults.	Find faults.
Secondary Goal	None.	Gather information, explain and evaluate contents, common understanding, establish consensus, education of reviewers. CN/B-06/3 Alternatives and stylistic issues discussed (see below).	Assess concepts and discuss alternatives, establish consistency, get feedback, inform participants.	Training.
Entry Criteria	None: document is presented in its current state (does not have to be finished).	Product and source documents distributed. Should be read but not formally reviewed. No formal criteria.	No formal criteria (document may be 50 - 80% complete). Kick-off very important: author can explain what (not) to review.	Complete document, every participant needs to review it and log defects. Different viewpoints.
Meeting	None.	Either to check	Logging and	No discussion during

	Informal	Walkthrough	Technical	Inspection
		direction of document or to explain contents to (usually) non technical people.	discussion not separated. Discussion necessary. Check only criticals and majors, keep track of people issues.	logging. Not everyone needs to remain for discussion phase.
Process Leader.	Author.	Moderator (often Test Manager).	Technical leader will act as moderator.	Moderator (often Test Manager).
Optimum team size.	2 (author and reviewer).	5 – 8.	4 – 6.	2 – 6 .
Preparation.	None.	Not extensive. Defects are found during the meeting. Roles are not needed.	Roles based on specific question of the author. Not too many pages.	Formal preparation using roles, based on rules, at a slow checking rate.
Materials Needed.	Document to review.	Document to review.	Document to review.	Document to review, source documents, standards, rules and checklists.
Participants	Anyone.	Peers, technical leaders and participants from outside the software discipline.	Experts (technical leaders), peers and sometimes project leaders.	Peers.
Logging.	Informal notes.	Separate scribe.	Author.	Author.
Exit Criteria	None.	No formal follow up as a later review is likely.	Author updates document. Technical leader verifies.	Moderator decides: document OK, or rework needed, or rework and meeting needed.
Follow up.	Informal.	Moderator checks whether action items are closed.	Expert-based decision regarding the follow up.	Formal check by the moderator and/or the participants.
Strengths	Quick and cheap.	Education of participants, sharing of knowledge, useful for non-technical staff.	Good for sharing knowledge with technical peers, making strategic technical decisions.	Very rigorous, best review method for finding defects.
Weaknesses	Finds less faults than other methods.	Finds less faults than other Technical Reviews and Inspections.	No reference to external documents.	Takes most time, most expensive.

	Informal	**Walkthrough**	**Technical**	**Inspection**
Applicable For	Unfinished documents or prior to other formal review with more people (especially customers).	Unfinished documents in early stages, to check if direction is desirable. And when document is (nearly) complete to get agreement (deployment).	Unfinished documents, to check technical contents.	Complete documents prior to acceptance.
Comments		Focus is only on document under review.	Aka Peer Review. Expert driven.	

MARK / TP/38.

6.3 Informal Review

Explain that Informal Reviews are simply one or more reviewers (other than the author) looking at a product and providing comments on it and that they rarely involve meetings.

Explain that there is normally no statement of objectives documented for an Informal Review and that they are characterised by there being no requirement for the results of the review to be documented.
Or archived.

6.4 Walkthrough

Explain that the product and source documents should be distributed to the reviewers so they can familiarise themselves with the product ahead of the meeting.
Documents should be read, (participants should be familiar with them) but not formally reviewed. MARK

Describe how the author also acts as the presenter and leads the reviewers through the product, often using scenarios for requirement and design models and dry runs through the code.

Explain that the secondary objectives, that distinguish Walkthroughs from other forms of review, are that alternatives and stylistic issues are examined and the education of reviewers is considered an important factor.

Summary:

- Author leads meeting.
- Other disciplines present.
- No formal preparation.
- Focus "only" on document under review.
- Different objectives (of participants, and of two different times when a walkthrough is held).
- Task scenarios and dry runs may be applied.

- Moderator is still important.
- Author is not the scribe.

CN/B-06/5

6.5 Technical Review

Explain that a Technical Review is also referred to as a Peer Review.

Describe how the product and source documents are distributed to the reviewers and the reviewers are expected to study the product ahead of the meeting.

Explain that a meeting is held where the reviewers report back on the document. The presenter at the meeting is not the author of the product. The presenter leads the meeting through the product sequentially (i.e. paragraph by paragraph). Reviewers discuss the issues raised and come to a consensus about what should be done next.

Explain that more reviewers may be involved in Technical Reviews than Walkthroughs or Inspections.

Summary:

- More expert driven.
- Earlier in the process.
- Kick-off defines the questions and roles.
- Focus "only" on document under review.
- Focus on content and technical approach.
- Discussion needed and allowed.
- Technical leader may act as moderator.
- Process and logging forms (different for each type of review).
- Very suitable for project management documents (test plan / project plan).

CN/B-06/3

6.6 Inspection

The Inspection method to be explained is based on Fagan Inspections.

Explain that the Inspection process is formally defined and followed rigorously. The outcome of the Inspection is based on whether the product meets pre-defined targets rather than consensus of the reviewers.

It is important to define quantifiable (entry and) exit-criteria:

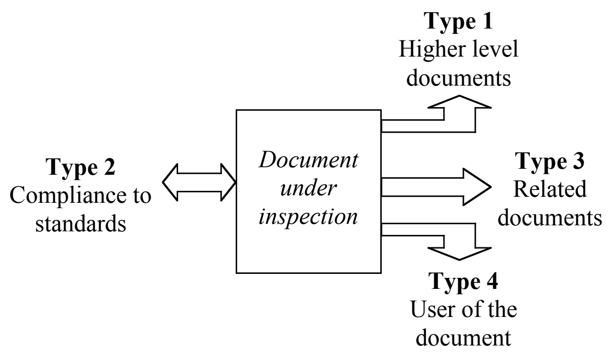
1. process: e.g. optimum checking rate
2. document: e.g. # Criticals / Majors per page < 3

BRYAN

Describe how an initial meeting is held so that all participants understand their responsibilities and the product and source documents. The time to be spent in individual checking is planned, and the parts of the product to be checked in more detail by a particular reviewer (also known as an inspector or checker) may be identified.

There are two kinds of forms: Review Process Form and Defect Logging Form
CN/B-04/1 + CN/B-05/1

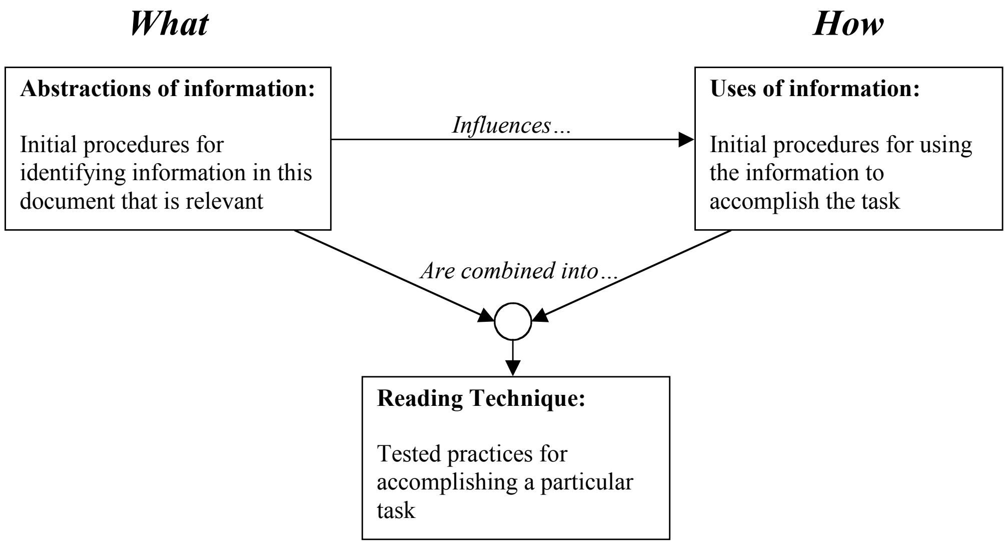
The initial meeting is often called the Kick-Off meeting.



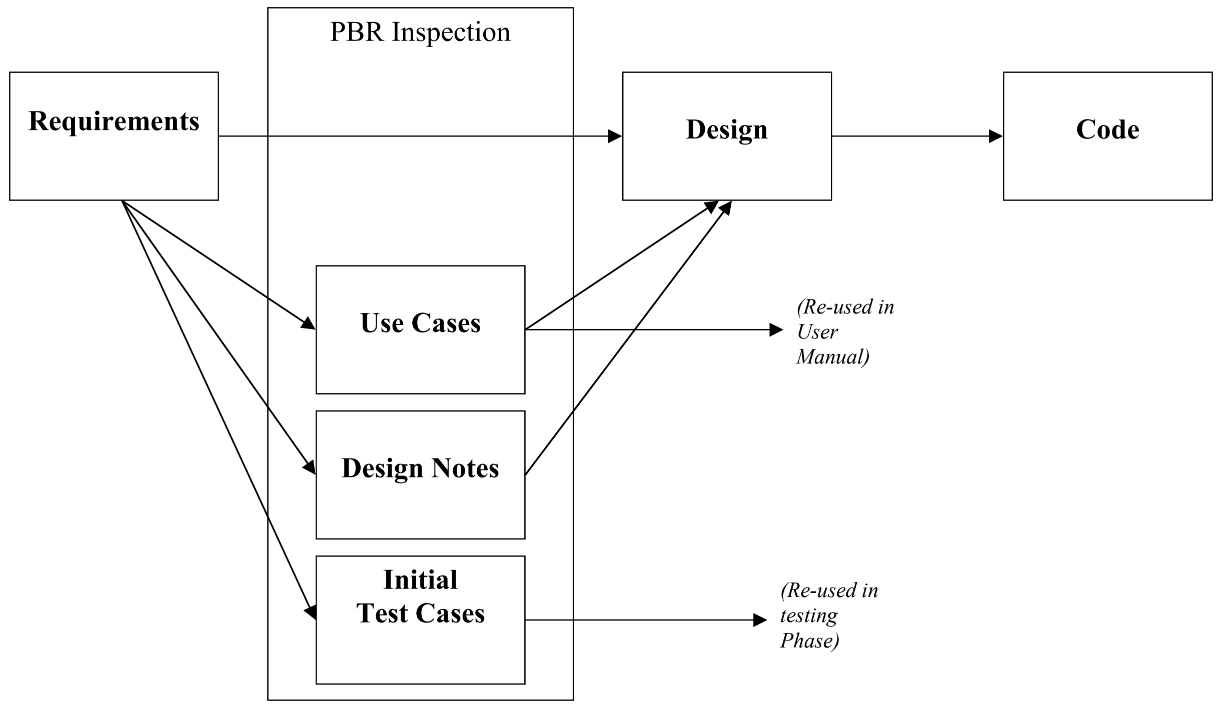
TP/124

The diagram above shows how each reviewer can have a different perspective, or focus.

Perspective Based Reading is explained in more detail by the diagrams below.


What
How
Abstractions of information:
Initial procedures for identifying information in this document that is relevant
Influences…
Uses of information:
Initial procedures for using the information to accomplish the task
Are combined into…
Reading Technique:
Tested practices for accomplishing a particular task

TP/147



TP/156

Explain that each reviewer inspects the product individually and reports back on the issues raised.

Describe how a logging meeting may be deemed necessary, dependent on the number of issues identified by the individual checking or the likelihood of finding additional issues during the logging meeting as predicted by measurements from previous inspections. The issues already found may be documented at this meeting, and new issues may also be identified. Discussion of whether a raised issue is, or is not, a fault is not permitted. No debugging is performed in the logging meeting.

Explain that there is a single leader for an Inspection, know as a Moderator or Inspection Leader, who plans the Inspection, including choosing the rest of the reviewers who make up the Inspection team. The Moderator also chairs any logging meeting and is responsible for ensuring that any raised issues are properly dealt with. The author is not allowed to present the product at a logging meeting.

Describe how the Moderator collects statistics about the time spent by each participant and the faults found. Estimates are calculated for the remaining faults per page, and the time/cost saved by the process.

Implementation steps:

- Management Support

- Provide people with some knowledge (presentation)
- Take a critical document to review
- Use experienced moderator the first time
- Train engineers and moderator
- Start inspecting
- Improve by using metrics and feedback sessions

CN/B-07/1

Stages of an inspection:

1. Planning: entry evaluation, define inspection strategy, gathering documents, invitation and planning
2. Kick-off: distribution of materials/documents, author introduces the document, moderator explains the (changed) process
3. Preparation: focus on role and assignment, check against rules and sources, focus on important defects, all documents
4. Meeting: logging (start, logging phase, decision phase (OK, moderator check, all participants check, new inspection), discussion, causal analysis
5. Rework, follow up, exit: rework (CR's on other docs, update doc by author, defect logging sheet as checklist, SPI proposals by moderator), follow up (check doc, doc under CM, report metrics), exit

CN/B-02/1 – CN/B-05/1

Return On Investment (ROI), indicates the benefit of the review

$ROI = (y_1 + (0.25 * y_2)) * z - x$

x = time spent in reviews (cost)
y_1 = number of criticals
y_2 = number of majors (1 in 4 leads to failure, hence the 0.25)
z = time to solve 1 fault found in testing

If the number is positive the review process has paid off.

When will reviews not work?
- Quality not important enough
- Management not interested
- Untrained engineers and moderators
- Information on document quality used by management to evaluate individual performance
- Inspections implemented in a too formal or theoretical manner
- Review/inspection remarks are given/taken personally

Success factors for a good review process:
- Find a champion, who will lead the process on a project or organizational level
- Pick things that really count, select documents for formal review that are the most important in a project

- Focus on role, but record all defects
- Checking not reading
- Respect preparation procedures
- Severity concept – critical, major, minor
- Explicitely plan and track formal review activities
- Follow the rules, keep it simple
- Respect optimum checking rates
- Collect data and provide feedback
- Continuously improve process and tools
- Perspective Based Reading
- Use a thorough entry check
- Report results
- Just do it!

MARIO

7 Incident Management

Describe and explain incident management as defined in IEEE Std. 1044-1993 Standards Classification for Software Anomalies and the supporting IEEE Std. 1044.1-1995 Guide to Classification for Software Anomalies.

Incident Management template should include:

- Unique identifier.
- Title.
- Date.
- Status:
  - New
  - Investigated
  - Resolved
  - Verified
  - Closed
  - Rejected.
- Description.
- Steps to reproduce.
- Test case.
- Tester name.
- Classification (Critical, Major, Minor, Textual).
- CR / PR (Change Request / Problem Report).

Advantages over Excel:

- Can assign status (force, e.g., investigated before resolved).
- Can determine access (e.g. only project manager may close).

Incidents need to be discussed in a CCB (Change Control Board). Here each item is discussed to determine:

1. Whether it is a defect / change to be implemented.
2. Its priority (e.g. Must, Should, Could or Won't (MoSCoW)).
3. When to be solved (this increment, next increment, postpone indefinitely).

MARK

Four Process Steps:
- Recognition
- Investigation
- Action
- Disposition

Each process steps has:
- Recording
- Classifying

- Identifying

CN/A-99/2

Steps:

a.	**Recognition**:	when an anomaly is found
b.	**Investigation**:	each anomaly is investigated to identify all known related issues and propose solutions
c.	**Action**:	a plan of action is formulated on the basis of the investigation
d.	**Disposition**:	once all actions required are complete the anomaly shall be closed

At each step there are administrative activities:

Recognition:

Recording:	who, what, where
Classifying:	project activity, suspected cause, repeatability, …
Identifying impact:	severity, priority, customer value, impact on schedule/cost/risk etc.

Investigation:

Recording:	effort, investigator name, start & end time
Classifying:	actual cause, source, type
Identifying impact:	review + update impact assessment from previous step

Action:

Recording:	items to be fixed, require action including corrective actions to prevent reoccurrence
Classifying:	resolution (immediate, eventual, deferred, no fix) and corrective (departmental, corporate, …)
Identifying impact:	review + update impact assessment from previous step

Disposition:

Recording:	action implemented, date closed, verification method (test case)
Classification:	closed, deferred, merged, referred to other project
Identifying impact:	review + update impact assessment from previous step

Above list taken from: IEEE 1044 – Standard classification for software anomalies
MARIO

8 Test Process Improvement

Explain that process improvement applies to both the software development process and to the testing process.

Provide an overview of the SEI Capability Maturity Model (CMMI) and ISO/IEC 15504 (SPICE) process improvement models. Make students aware of the SEI CMMI and briefly describe its relationship to the SEI CMM and ISO/IEC 15504.

Describe, in detail, the Testing Maturity Model (TMM, as defined by IIT) and TPI®.

TMM

TMM is based on CMM(-i). Also has five maturity levels:

1. Initial.
2. Definition.
3. Integration.
4. Management and Measurement.
5. Optimisation.

Most organisations are at level 1.
TP/45 and MARK.

TMM Level 2 consists of: Test policy and goals, Test planning, Test techniques and methods, Test environment
CN/D-02/7

Objective of testing are different for the TMM levels:

TMM L1:	show that software "runs"
TMM L2:	show that software meets its specification
TMM L3:	provide insight into the quality of the product, e.g. outstanding risks
TMM L4:	quality measurement (evaluation)
TMM L5:	prevent defects from re-occurring in the future

CN/D-02/6-8

TPI

TPI involves tracking 20 KPAs (Key Process Areas) at up to four levels of maturity across 13 project levels in a Test Maturity Matrix.

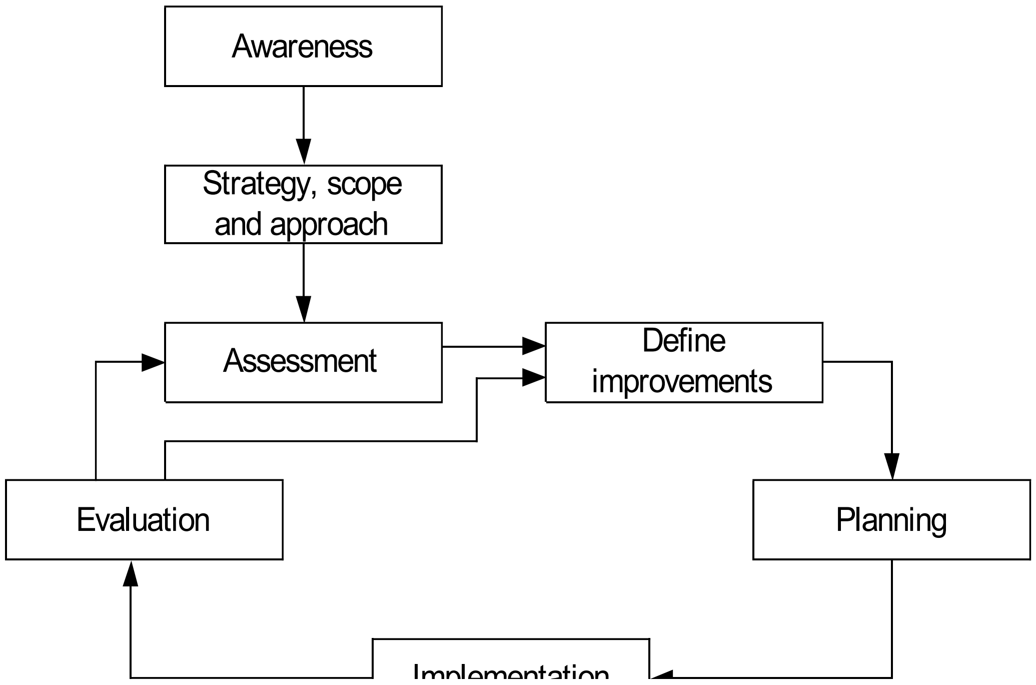
Pre-requisites for (S|T)PI:

- Management awareness
- Resources
- Realistic targets (t (time), € (money), Q (Quality))
- Focus not only on process, but also on change management
- Commitment on all organisation levels

CN/D-01/5

TMM vs.	TPI
CMM(-I) related	TMap related
All test levels (incl. static testing)	Focussed on higher test levels
Testing=evaluation	Testware engineering (practical)
Top-down model	Bottom-up model
Management commitment	Test engineering (practical)
Highly focussed	20 key areas (a lot!)
Detailed	Highly detailed (start right away)
Research product	Commercial product (foundation??)
Books and guidelines	Book in public domain
5 levels	13 scales of maturity
Staged model	Continuous model
Higher maturity (TMML4 and TMML5)	Lower maturity

CN/D-02/10 + BRYAN



CN/TPI-Reader/14

9 Test Tools

Note: this section is so detailed in the syllabus that it seems impossible to add further comments! MARK

9.1 Overview

Explain that tool support is available for many testing activities throughout the system lifecycle. Describe each of the following types of tool and how they are used and explain the benefits and pitfalls of each:

- *Requirements testing tools provide automated support for the verification and validation of requirements models, such as consistency checking and animation.*
- *Static analysis tools provide information about the quality of the software by examining the code, rather than by running test cases through the code. Static analysis tools usually give objective measurements of various characteristics of the software, such as the cyclomatic complexity measure and other quality metrics.*
- *Test design tools generate test inputs from a specification that may be held in a CASE tool repository or from formally specified requirements held in the tool itself. Some tools generate test inputs from an analysis of the code.*
- *Test input data preparation tools enable data to be selected from existing databases or created, generated, manipulated and edited for use in tests. The most sophisticated tools can deal with a range of file and database formats.*
- *Test running tools provide test capture and reply facilities. The tools capture user-entered terminal keystrokes and capture screen responses for later comparison. For Graphical User Interface (GUI) applications the tools can simulate mouse movement and button clicks and can recognise GUI objects such as windows, fields, buttons and other controls. Object states and bitmap images can be captured for later comparison. Test procedures are normally captured in a programmable script language. When the script is executed this replays the user-entered keystrokes, etc. and can compare actual responses with those captured previously. Data, test cases and expected results may be held in separate test repositories. These tools are most often used to automate regression testing.*

Advantages of automation:

- Tools can do boring jobs leaving testers to do more interesting work.
- Tools do not make mistakes (though the testers who programmed them may have done).
- Tests are easy to repeat (simply run tool).
- Tests can be run outside of office hours.

Disadvantages of automation:

- Cost of purchase: the tools are not cheap.
- Maintenance can be a lot of work (certainly basic tools).
- Difficult for non-programmers to maintain (adjusting scripts).
- Takes time to learn and get maximum benefit from.
- Risk of becoming shelfware (see 9.2 "Tool Selection").

MARK

1. Record & Playback

2. Test Programming
   (split data & test flow)
3. Data Driven Testing
   (test case in separate file and function library)
4. Keyword Driven Testing
   (action words)

CN/C-13/3-6

- *<u>Test harnesses and drives</u> are used to execute software under test which may not have a user interface or to run groups of existing automated test scripts which can be controlled by the tester. Some commercially available tools exist, but custom-written programs also fall into this category.*
- *<u>Test script generators</u> generate test scripts from test specifications.*
- *<u>Simulators</u> are used to support tests where code or other systems are either unavailable or impracticable to use (e.g. testing software to cope with nuclear meltdowns).*
- *<u>Performance test</u> tools have two main facilities: load generation and test transaction measurement. Load generation can simulate either multiple users or high volumes of input data. Load generation is done either by driving the application using its user interface or by test drivers, which simulate the load generated by the application. Records of the numbers of transactions executed are logged. Driving the application using its user interface, response time measurements are taken for selected transactions and these are logged. Performance testing tools normally provide reports based on test logs and graphs of load against response times.*

  Load generation: e.g. multiple users, high volumes
  Test transition measurement: e.g. network traffic, CPU usage, memory usage

  Reference: CN/C-12/5

- *<u>Dynamic analysis</u> tools provide run-time information on the state of executing software. These tools are most commonly used to*
  - *Identify unassigned pointers,*
  - *Check point arithmetic,*
  - *Monitor the allocation, use and de-allocation of memory to flag memory leaks and*
  - *Highlight other errors difficult to find 'statically'.*
- *<u>Debugging</u> tools are mainly used by programmers to reproduce faults and investigate the state of programs. Debuggers enable programmers to execute programs line by line, to halt the program at any program statement and to set and examine program variables. It should be made clear that debugging (and debugging tools) are related to testing but are not testing (or testing tools).*
- *<u>Comparison</u> tools are used to detect differences between actual results and expected results. Standalone comparison tools normally deal with a range of file or database formats. Test running tools usually have built-in comparators that deal with character screens, GUI*

objects or bitmap images. These tools often have filtering or masking capabilities, whereby they can 'ignore' rows or columns of data or areas on screens.

- *Test management tools may have several capabilities. Testware management is concerned with the creation, management and control of test documentation, e.g. test plans, specifications and results. Some tools support the project management aspects of testing, for example the scheduling of tests, the logging of results and the management of incidents raised during testing. Incident management tools (also known as defect tracking tools) may also have workflow-oriented facilities to track and control the allocation, correction and retesting of incidents. Traceability tools allow the link between test cases and their corresponding test coverage items to be recorded. Most test management tools provide extensive reporting and analysis facilities.*
- *Coverage measurement (or analysis) tools provide objective measures of structural test coverage when tests are executed. Programs to be tested are instrumented before compilation. Instrumentation code dynamically captures the coverage data in a log file and necessarily slows the program, which may affect the functionality of the program under test. After execution, the log file is analysed and coverage statistics generated. Most tools provide statistics on the most common coverage measures such as statement or branch coverage.*
- *Hyperlink testing tools are used to check that no broken hyperlinks are present on a web site.*
- *Monitoring tools are typically used for testing e-commerce and e-business applications as well as web sites. The main function of this tools is to monitor web sites to ensure they are available to customers and to produce a warning if the service begins to degrade or has failed.*
- *Security testing tools are typically used for testing e-commerce and e-business applications as well as web sites. A security testing tool will check for any aspects of a web based system that are vulnerable to abuse by unauthorised access.*
- *Test oracles are generally used automatically to generate expected results. As such they perform the same function as the software under test and so are rarely available. They may be used, however, in situations where an old system is being replaced by a new system with the same functionality, and so the old system can be used as an oracle. Oracles may also be used where performance is an issue for the delivered system. A low performance oracle may be build/used to generate expected results for the functional testing of the high performance software to be delivered.*

9.2 Tool Selection

Explain the importance of a formal tools evaluation and selection process to avoid buying an inappropriate or unnecessary tool, so that purchased tools are less likely to end up as shelfware (unused).

This is an entire project.
CN/D-04/3

Describe how the tool evaluation and selection process ideally comprises the following activities:

- *Identify and quantify the problem;*
- *Consider alternative solutions;*
- *Prepare a business case;*
- *Identify constraints;*
- *Identify required tool features and characteristics;*
- *Prepare a short-list of candidate tools;*
- *Undertake a more detailed evaluation;*
- *Perform a competitive trial, if necessary.*

Also consider supplier presentation. Then select tool and perform implementation (pilot project).
CN/D-04/4

Explain that the evaluation and selection team typically comprises a full time team leader with part time team members representing different areas of the organisation where the tool may be used.

9.3 Tool Implementation

Explain the importance of a formal tool implementation process as a means of avoiding failure of test automation in the long term.

People issues (holds for tool implementation but also for (S|T)PI):
f(a,b,c) > z, where:
a: dissatisfaction with current state
b: shared vision of the future
c: concrete knowledge about the steps to get from a to b
z: the psychological or emotional cost to the individual of changing the way they work"
CN/TPI-Reader/12

Describe how the implementation process starts with a small-scale pilot project to verify the business case and establish a standard approach to using the tool within the organisation. This approach should include identification of a testware architecture, scripting techniques, naming conventions and configuration management of testware, as appropriate. The implementation team should ideally work full time on the pilot project that would typically be between 3 and 6 months duration. Team members may undertake specific roles including the following:

- *<u>Champion</u> – the driving force behind the day to day implementation of tool support, understands the issues involved and is enthusiastic about the potential benefits, is able to work well with people;*
- *<u>Change Agent</u> – plans and manages the implementation of the tool (including the pilot project). This person may also be the Champion;*
- *<u>Tool Custodian</u> – responsible for technical tool support, providing internal help or consultancy in the use of the tool.*

Explain that in addition to these roles and the work of other team members there is ideally a Management Sponsor, a senior manager who visibly supports the implementation of a testing tool.

Describe how, at the end of the pilot project, the results are assessed against the business case and if successful the use of the tool is progressively rolled out to other projects and teams using the approach developed during the pilot project.

10 People Skills

10.1 Individual Skills

Explain that an individual's testing capability can be derived from experience and/or training in one or more of the following areas: users, development and testing. Users are familiar with the application from the user's perspective and this provides insight into how the system will be used and where failures would have the greatest impact. In some cases, in depth knowledge of the application domain may not reside with (naïve) users, but still provides a useful insight for the testing of a system. Knowledge of development skills (requirements analysis, design and coding) gives insight into how possible errors are made and the possible faults that could be introduced. More specific testing skills include the ability to analyse a specification, design test cases and the diligence for running and checking tests.

Explain that interpersonal skills, such as giving and receiving criticism, influencing and negotiation are all important in the role of testing.

Balance: technical skills, soft skills, attitude
CN/A-14/3

Testing skills:
- Test knowledge
- Domain knowledge
- IT knowledge
- Social skills

MARIO

Soft skills and attitude:
- communication (oral, reporting)
- diplomacy
- giving and receiving constructive criticism / positive feedback
- influencing people
- negotiation
- selling/promoting testing
- stress resistant
- be accurate
- sense of humour

CN/A-14/2

Test knowledge	**IT knowledge**
- test principles - techniques - tools	- software - requirements - CASE tooling
Domain knowledge - business process - user characteristics	**Social skills** - communication - presentation - accurate - reporting - diplomacy

CN/A-14/1

10.2 Test Team Dynamics

Present the issues of staff selection, such as ensuring that a new recruit complements the skills and personality types that already exist within the test team. Present the advantages of having a variety of personality types within the test team.

Based on the work of Dr Meredith Belbin, introduce the concept of a team role as the way someone behaves, contributes and inter-relates when working in a team.

Explain that there are a finite number of team roles which comprise certain patterns of behaviour which can be adopted naturally by the various personality types found among people at work. Describe these roles and explain how they apply to test teams.

Belbin Team Roles:

Role	**Advantage**	**Disadvantage**
Coordinator	Organiser	Manipulative
Shaper	Driver	Temperamental
Resource Investigator	Bridge Builder	Over Run
Monitor Evaluator	Methodical	Negative
Implementer	Disciplined	Inflexible
Completer Finisher	Focused	Obsessive
Specialist	Expert	Closed
Team-worker	Popular	Indecisive
Plant	Innovative	Sensitive

CN/A-14/4

10.3 Fitting Testing within an Organisation

Explain that organisations may have different organisational structures for testing:

- *Testing may be the developer's responsibility,*
- *Or may be the development team's responsibility,*
- *Or one person in the development team is the tester,*
- *Or there is a dedicated test team (who do no development),*
- *Or there are internal test consultants providing advice to projects,*
- *Or a separate organisation performs the testing.*

Describe the different forms of communication, to include that between:

- *Testers and Developers. Testers find faults in the developer's product, but this needs to be conveyed diplomatically as a means of improving the quality of the product. Developers should inform the testers which areas of the software may need particular focus, such as highly-complex or new software.*
- *Testers and Project Management. Testers report on the progress of the testing and on the quality of the software to project management.*
- *Testers and Users. Testers receive information on areas that are most important to users, allowing them to prioritise testing. Testers receive background information on the application area. Users will often receive or see test results, but those with little knowledge of testing may need help in the interpretation of these results.*

10.4 Motivation

Cover motivating factors, such as recognition and respect. Cover demotivating factors, such as the lack of career paths.

Recognition and respect are gained by the testers noticeably providing added value to the project. On a single project testers may achieve this most quickly by participating in the early reviews. Over a period of time testers will gain recognition and respect from their record of adding value on previous projects.

Job satisfaction:

- adding value
- receiving recognition
- gaining respect
- feeling valued

Career paths

- job description, what's my role?
- training (learn & progress)
- promotion (chosen career)
- agreed targets ind./team performance

CN/A-14/9

Appendix A: V-Model Table

	Component Test	**Integration In The Small**	**System Test**	**Integration In The Large**	**Acceptance Test**
Objective	To show that each component functions as intended (according to the component specification) and to provide visibility into the quality of the component	To confirm functionality of modules when components are combined together. Verification against the global design, with focus on interaction and interfacing between modules.	To show that the system meets the functional specification and all the non-functional requirements.	To confirm system and network functionality when the system is integrated into an existing network of systems.	To show the delivered system meets the business needs (validation), formal acceptance of the product
Scope	Test Input Fields, GUIs, code, calculations, etc	Test all interfaces, data exchange, and variable passing between components.	Functionality, non-functional attributes such as performance, security, installation, error handling, recovery etc	System interfaces, files, data, operational profiles, usually end to end testing based on business processes.	Requirements based testing. The whole system, test cases based on requirements document, covers functionality, usability, help, user guides etc.
Responsibility	Development	Integration team	Test team (within development)	Test team / Integrators.	User / Customer or representative
Who does it	Developer typically, development owned	Integration tester, integrator / development responsibility	Independent Test team, technical experts / development responsibility	Independent test team	Users, business representatives, live support, or test (user/customer responsibility)
Entry Criteria	Component is complete and ready to test (compiles, has been reviewed, software design has been approved, complies to static criteria).	Components passed component test phase (complies to exit criteria), integration test spec reviewed and approved, global design reviewed and approved, passed confidence test.	Integration test phase passed, development sign off, test team acceptance (intake/ confidence), release notes available	System test sign off for each system being integrated	Sign off from System Test/ Integration test phase (system test report available), user requirements reviewed and approved, test plan reviewed and approved.
Exit Criteria	Component passes all tests and is signed off, level of coverage is reached, test report written, cyclomatic complexity within criteria	All integration test cases executed, all critical tests passed, number of defects within set limits, test results documented, test report written	All system test cases run and complete, no high priority defects outstanding, Mean Time Between Failure (MTBF), # defects per test hour under threshold, requirements coverage	All integration test cases complete, no category A or B bugs outstanding, system meets reliability requirements	All acceptance test cases completed, no category A or B business priority defects outstanding, list with known defects, business sign off for live implementation, MTBF, acceptance test report approved
Test Deliverables	Component test report/results, test log, test code & test data, component sign off of developer/ stubs and drivers.	Integration test report/result, problem reports, sign off, integration test specification	System test plan, test report, test results, test specifications and – procedures, test evaluation report (recommendations for product and project)	Integration test report, test results, configuration guide, integrated test model	Acceptance test report, results, test logs, problem reports, change requests, test specifications and test procedures.
Typical Test Techniques	White box coverage techniques, syntax testing, Boundary Value Analysis, Ad hoc	White box test techniques, equivalence partitioning, state transition testing, syntax, cause effect graphing.	Black Box (e.g. Equivalence Partitioning, state transition, cause/effect graphing), specialist non-functional test techniques (e.g. error-guessing)	Black Box	Equivalence partitioning, exploratory testing, use case testing, error guessing, process cycle test
Metrics	Number of defects found/fixed, priority, statistical analysis	Number of faults found, priority, statistical analysis	Tests passed/failed/run, number of faults and priority, environment log, test logs,	Test passed/failed/run, number of faults and priority test coverage, test effectiveness	Number of faults, business priority, tests passed/failed/run

	Component Test	**Integration In The Small**	**System Test**	**Integration In The Large**	**Acceptance Test**
			progress reports time & effort planned v spent, requirements coverage, test effectiveness		
Test Tools	Static analysis tools, debuggers, harnesses, dynamic analysis, coverage, comparator	Test harnesses (stubs, drivers), simulators, record/playback for regression testing, defect management, test management	Performance monitoring, data generators, capture/replay, test management tools	Harnesses, stubs, drivers, data generators, simulators, capture/replay, comparators	Capture replay, comparators, performance, defect management, test management
Applicable Testing Standard	BS7925-2 (component testing standard, TMap, MISRA coding standards	BS7925-2 (component testing standard, TMap, IEEE 829 for documentation	BS7925-2 (component testing standard, TMap, IEEE 829 for documentation, ISO 9126 for non-functional exit criteria	See useful standards for software testing	BS7925-2 (component testing standard, TMap, IEEE 829 for documentation, ISO 9126 for non-functional exit criteria
Typical non functional test types	Resource usage (e.g. memory, semaphores, …), time behaviour, portability, maintainability	Resource usage (e.g. memory), performance	Reliability, performance, usability, portability		Usability, performance, security

Appendix B: Exam Tips

B.1 Tips From ISEB For The Practitioner Exam

The following tips come from ISEB themselves. They can be found at this site:
http://www.bcs.org/BCS/Products/Qualifications/ISEB/Areas/SoftTest/practitionerexamguidance.htm

Guidance to Practitioner Examination Candidates

With the experience of 3 examinations behind us it is clear that not all candidates for the Software Testing Practitioner Certificate examination have been well prepared. The purpose of this note is to assist course providers in briefing their delegates on what to expect in the examination and to help candidates to avoid some common pitfalls.

- *The Examination*
- *Common Problems*
- *Planned Revision*
- *Advice to Course Providers*

The Examination

The Software Testing Practitioner Certificate examination is a 3 hour written paper consisting of a single compulsory question (worth 40% of the marks) and 5 optional questions (each worth 20% of the marks) from which 3 questions must be attempted.
The examination attempts to mirror the syllabus in that questions are set from syllabus topics. A balance across the syllabus topics is sought in each individual examination paper and also between papers. Questions attempt to reflect the requirement that a significant component of taught courses should be practical in nature.
In short, the examination sets out to identify whether a candidate has acquired the knowledge defined in the syllabus and whether the candidate can apply that knowledge in a practical setting.

Common Problems:

Candidates taking the examination have tended to make similar mistakes, many of which are easily remedied with a little application. The most common problems are listed here for your guidance:

- ***Not answering the question set.*** *Examination questions usually incorporate a simple scenario that sets the context for the question. Candidates who do not pay enough attention to this information tend to answer the question 'write down all you know about.....' rather than addressing the specific issue(s) raised by the question. While some of the candidate's answer may be relevant, there is a serious risk that much of it will not be and will earn no marks.*
- ***Answers not thought through.*** *Candidates need to structure their response to a question to ensure that they impart the required information with minimum effort. Long and rambling answers run the risk of overlooking key information and the relative importance of these ideas may be undermine by the lack of structure. Long answers also take longer to write down than a well structured but shorter response.*
- ***Answers are too superficial.*** *Questions are worded to indicate the level of response required. Sometimes a word count is included. If a question asks for a description of something this will*

require more than simply identifying the something by name. Similarly 'justify' means provide some kind of evidence that your response is correct or appropriate. All information in an answer must be relevant; any irrelevant information will not earn marks. The candidate therefore wastes valuable time in writing down irrelevant information.

- ***Addressing the scenario.*** *Where a scenario is provided (especially in double length questions) there is an expectation that candidates will refer to the scenario in their answer, using it to identify specific examples of points raised or to illustrate issues. In a Practitioner exam it is important to demonstrate practical validity of answers wherever possible.*
- ***Running out of time.*** *The problems mentioned all contribute to wasted time. There is time to construct and write down good answers to the questions on the paper if time is not wasted. A good time plan is essential and we recommend that every candidate takes 15 minutes to read and understand all the questions before selecting the most promising ones to answer. A further 5 minutes should be allowed for checking the question carefully before setting out a response. It is a good idea to set out the key points of an answer in 'bullet point' form before starting to draft the text of the answer. At the end of the paper further time should be allowed for checking through the answer paper to ensure that no major mistakes have been left uncorrected.*
- ***Illegible answers.*** *It is difficult to write both quickly and neatly. However, an illegible scrawl makes it very difficult for a script marker to read a candidate's response and important information may be unintelligible from the script. Time planning should reduce the need to rush. Practice in writing at more than the usual speed would be sensible preparation for the examination, especially for candidates who do not normally communicate with pen and ink. You may find it helpful to highlight key points in your answer by underlining or writing in block capitals.*

Planned Revision

Candidates preparing for the examination should revise the syllabus topics carefully. An ideal revision programme would be structured around the syllabus with revision time allocated in proportion to the time allocation for a topic in the syllabus.
A sample examination paper is available from ISEB to provide guidance on the form of the examination and the nature of the questions set.

Advice to Course Providers

Course providers can maximise their candidates' chances of passing the exam by paying attention to some basic principles:

- *Exams are set from the syllabus and are, as far as possible, balanced by topic and level. The level of coverage identified in the syllabus is a guide to the level of understanding that could be expected in an exam question.*
- *Exam technique is a major factor and course providers should ensure that candidates are at least briefed on the guidelines offered elsewhere in this document.*
- *An element of 'testing common sense' is expected in candidates in answering questions. A candidate with 18 months' experience is the model, so course providers need to ensure that less experienced candidates are given enough practical training to enable them to answer exam questions from a practical perspective.*
- *Candidates should be able to answer a 'compare and contrast' question by setting out the relevant aspects of the items to be compared and then writing a succinct conclusion based on the comparative information presented.*

- *In white box questions it is expected that candidates will be able to read and analyse pseudo code; this may be written to look like 'real' code i.e. with declarations and punctuation. Candidates should be capable of doing simple code analysis and generating test cases from pseudo code. Statement and branch testing should be understood well enough to create test cases.*
- *In black box questions candidates will be expected to extract information from a scenario and generate test cases. Equivalence partitioning, boundary value analysis, state transition testing and classification tree method should be well enough understood to be able to write test cases.*
- *Some aspects of the syllabus have not, so far, appeared frequently (or at all) in exam papers. This is not a reliable guide to exam content, and all areas of the syllabus are regarded as examinable.*

Course providers are asked to make the content of this guidance note available to their delegates at some point during course presentation, together with some advice on exam preparation

B.2 Further Tips

The following tips are gleaned from other sources, including my own experience of the exam.

- Study the syllabus and memorise points / lists if possible (hence this document and the abbreviations).
- Assume that you will run out of time and focus on getting the most marks in the minimum time (so leave a question unfinished if you have spent too long on it and come back to it later if possible).
- Consider points per minute. Take into account that you have 3 hours, minus 15 minutes for reading and selecting your questions and another 15 minutes for a final review. That makes 1.5 minutes per point to gain. So, for a 6 point question you have 9 minutes. It might feel hard to stick to, but it is worth it. It is better to go to a new question and try to get half the points of that (e.g. 3 on 6), than continuing your current question for that last point you <u>might</u> get.
- Have a few topics learnt well and skim the others if time is short for preparation, rather than trying to learn, "a little bit of everything". This should ensure that you can perform well on at least two (or hopefully three) of the optional questions.
- Do practice questions.
- Practice writing by hand: you will be writing almost continuously for three hours!
- Use a highlighter pen to highlight the important parts of a question on the paper itself (e.g. details of the scenario, like "new company" or "experienced test team"). Alternatively, take two pens: an extra one for instructions to you (e.g. "list" or "describe").
- Neat handwriting may help put examiner in a better mood.
- Use a ruler to improve neatness when drawing diagrams (note – this can be time consuming).
- Do not start writing immediately. First read the question entirely (using highlighters) and then make a few notes / bullet points first, identifying for which part of the question you will write which answer. This can be done on the question paper.
- Food can be taken to the exam, as long as it is not noisy (so no crisps or apples!). Be warned: food can send you into a dip. I took a fruit salad (strawberries and oranges) hoping that it would stimulate my brain!

- You are allowed to leave and go to the toilet, one at a time. I found that just a few minutes out, with a bit of movement walking outside, helped to re-energise me.

B.3 Exam Essentials

The following points are not hints but essentials:

- You must use a **black pen** in the examination. (Nobody seems to know why, and it is not mentioned in any of the documentation that I have seen.) If you do not take one, the examiner will provide one (but it is a good idea to take one with which you are familiar – for the three hours of writing!).
- You must have your **candidate number** with you. This is not the same as the one for the Foundation Exam. Your course provider will give you your candidate number, but make sure you take it to the exam.
- On each sheet of paper you use, you will need to write three things:
  - Your candidate number.
  - The question number.
  - The page number. Note that this is **per question**, so write 1, 2, 3, … for the first question and begin again at 1 for the next question.

Appendix C: Useful Websites

Site	Description
http://www.geocities.com/robinson_weijman	Latest version of this document.
http://groups.yahoo.com/group/IsebCertification	This group is intended for everyone who is interested in ISEB Software Testing Certification.
http://www30.brinkster.com/wvole/#hum_error	A long list of useful testing links.
http://www.bcs.org/BCS/Products/Qualifications/ISEB/Areas/SoftTest	The ISEB Software Testing page.
http://www.sogeti.nl/tpi	Page devoted to TPI with tool downloads.
http://www5.in.tum.de/~huckle/bugse.html http://www.cs.tau.ac.il/~nachumd/verify/horror.html	These sites contain annecdotes of when software was not well tested. Useful for spicing up articles / speeches on software testing.

Appendix D: Summary Of Test Techniques

D.1 Introduction

This chapter gives a brief introduction to various test techniques and explains when they may be used. The purpose here is not to enable testers to use these techniques, but simply to be made aware of them and when they might be used. When a tester encounters a situation in which one of these techniques could be used, *then* the technique can be learnt.

Further information can be found in this document: *Standard for Software Component Testing, Working Draft 3.4, 27 April 2001* produced by the *British Computer Society Specialist Interest Group in Software Testing (BCS SIGIST)*. It can be found at http://www.testingstandards.co.uk/

Some test techniques are similar but are different "strengths". E.g. for a high-risk component, exploratory testing could be used, for medium-risk error guessing could be used and for low-risk ad hoc testing could be used.

Test techniques that have not been included here are: document reviews (Walkthrough, Inspection, Peer (aka Technical) and Informal), Complexity Analysis, Condition Testing (Branch Condition, Modified Condition Decision and Branch Condition Combination) and LCSAJ Testing.

D.2 Equivalence Partitioning

What is it?

Equivalence Partitioning is a test technique in which inputs and outputs that display similar behaviour are grouped together. Thus, by testing one input or output from a group, it is assumed that the whole group is tested. The groups are called, "Equivalence Classes".

Example

If an exam has a 60% pass rate, then the scores 60 – 100% are assumed to be equivalent: testing any one of them tests the whole group. E.g. it is not necessary to test 70%, 80% and 90%.

When is it used?

Whenever there are inputs or outputs in a range. It enables test cases to be objectively created. Ideally used in combination with Boundary Value Analysis.

D.3 Random Testing

What is it?

Random testing is when certain inputs are possible but Equivalence Partitioning is not practised. Rather, inputs are chosen at random.

Example

Same as Equivalence Partitioning example, but inputs can be anything, e.g. 25%, 56%, 91%.

When is it used?

Whenever Equivalence Partitioning is used but no time is available to analyse the scenario to create Equivalence Classes. Can be used in conjunction with a tool that generates random inputs.

D.4 Boundary Value Analysis

What is it?
Having partitioned inputs and outputs (see "Equivalence Partitioning"), this test technique tests the *boundaries*. This is because defects are more likely to occur at the boundaries of equivalence classes than at any other point.

Example
If an exam has a 60% pass rate, then it is best to test the following values: 59% (fail), 60% (pass) and 61% (pass). However, the equivalence classes also contain other boundaries, so it is also useful to test: -1% (invalid), 0% (fail), 1% (fail), 99% (pass), 100% (pass) and 101% (invalid).

When is it used?
Whenever there are inputs or outputs in a range. It enables test cases to be objectively created. Requires an understanding of Equivalence Partitioning.

D.5 State Transition Testing

What is it?
This test technique involves testing a system as it changes from one state to another. The initial state, input, output and final states are defined. Note that this technique is easily scalable, so that for a high-risk component more thorough testing can be achieved by testing the transition from the initial state to the final state via an intermediate state (or more than one intermediate state).

Example
A digital watch, which can exist in four possible states: *display time*, *change time*, *display date* and *change date*. Changing from *display time* to *change time*, the input would be "press reset" and the output would be "alter time".

When is it used?
Whenever a component can exist in certain states.

D.6 Cause Effect Graphing

What is it?
This technique involves analysing the causes and effects on a component. The word "Graphing" is misleading: graphs can be created but are rarely done so because the test cases can more easily be created via a table, called a *decision table*.

Example
A company decides to mailshot everyone in its database. The type of mail sent depends on the following "causes": age of customer and gender of customer. The "effects" are different types of mailshots.

When is it used?
Useful whenever a scenario involves combinations of causes and effects.

D.7 Syntax Testing

What is it?
Testing input via syntax.

Example
If a field requires the age of a customer, valid entries would be an integer (possibly within a range). Invalid entries would be: letters, decimals, special characters (€, @, or TAB) or nothing (leave field blank).

When is it used?
Whenever an input requires a certain syntax. Can also be used to test interfaces between components (integration testing).

D.8 Statement Testing

What is it?
Statement testing involves running a program and comparing the outcome with the code.

Example
For code "if *a* then *b*", *a* should be true to test all the code (i.e. also test *b*).

When is it used?
This technique can be used for any program. It is best used in conjunction with a tool that can measure the coverage and highlight code that has not been tested. Compare with Branch / Decision Testing.

D.9 Branch / Decision Testing

What is it?
Branch Testing and Decision Testing are very similar. Here they will be treated as identical. Branch / Decision Testing involves checking paths through the code.

Example
For code "if *a* then *b*", two tests should be run for 100% coverage: one with *a* true and one with *a* false.

When is it used?
As with Statement Testing, it can be used for any program but is best used in conjunction with a tool to measure coverage.

D.10 Data Flow Testing

What is it?
This technique focuses on how variables are used within code. A path is traced from where a variable is initialised to where it is used.

Example
Can be used with any code. Simply select one variable and watch how it changes as the code is executed.

When is it used?
This is a useful technique for developers to perform their own tests since they can observe this easily using debugging tools.

D.11 Ad Hoc Testing

What is it?

The tester simply tries things at random. The minimum knowledge is to have an understanding of the requirements in order to compare expected and actual outcome.

Example

For an online program in which users need to log in and enter personal information, a tester will simply try that a few times with different information entered into different fields.

When is it used?

All too frequently! More seriously, this technique will find the most serious defect in the least amount of time. The overhead of writing test cases is gone. This is useful for "Friday afternoon" testing.

D.12 Error Guessing

What is it?

An experienced tester has an intuitive idea of where bugs can be found. Error Guessing is similar to Ad Hoc Testing except that the tester first tries to consider where defects might lie. They may also attempt to test the most error prone or most visible components first.

Example

For the online program described under, "Ad Hoc Testing", a tester might begin with standard input and then try non-standard input, e.g. informal syntax testing, entering a non-valid postcode or credit card details, etc.

When is it used?

Error Guessing is used when one experienced tester is given a product and simply asked to, "test it". Often the project will be under time pressure and so speed has a higher priority than test quality. (Any measure of quality of the component will be subjective.)

D.13 Exploratory Testing

What is it?

This is similar to Ad Hoc Testing and Error Guessing, but is more in-depth than either. Exploratory Testing begins with a meeting between two experienced testers and a manager. The group focuses on identifying risk prone areas and then considers what type of testing would be appropriate. The testers then test the product and report back. This procedure is repeated once a day or so, with one meeting per day. In a meeting the manager will typically ask, "What is the most interesting or important defect you have found today?"

Example

For the online program described under, "Ad Hoc Testing", the team might decide that particularly risky items include: leap year dates, international address formats, unusual credit cards and foreign characters (like "ô").

When is it used?

Used in similar conditions to Error Guessing but on more risky components.

Appendix E: List Of Standards

BS7925-1	: Software Testing Vocabulary
BS7925-2	: Software Component Testing
CMM-SW	: Capability Maturity Model for Software
CMMI-SE/SW	: Capability Maturity Model Integration
IEEE 610	: Standard Computer Dictionary
IEEE 610.12	: Software Engineering Terminology
IEEE 730	: Standard for Software Quality Assurance Plans
IEEE 829	: Standard for Software Test Documentation
IEEE 1008	: Standard for Software Unit Testing
IEEE 1012	: Standard for Software Verification and Validation
IEEE 1028	: Standard for Software Reviews
IEEE 1044	: Standard for Classification for Software Anomalies
IEEE 1044.1	: Guide to Classification for Software Anomalies
TMM	: Testing Maturity Model