

INFORMATION SYSTEMS EXAMINATIONS BOARD

Practitioner Certificate in Software Testing

Guidelines & Syllabus

Version 1.1 – 4th September 2001

Background

This document is the syllabus for the second level of qualification in software testing, as administered by the British Computer Society's Information Systems Examination Board (ISEB). Background information on ISEB and the first level qualification, the Foundation Certificate, may be found at www.bcs.org.uk/iseb.

A testing practitioner is anyone involved in software testing. This includes testers, test analysts, test engineers, test consultants, test managers, user acceptance testers, and software developers. The Practitioner Certificate is intended to be a qualification appropriate to all testing practitioners.

Entry Criteria

The entry criteria for candidates taking the ISEB Practitioner Certificate in Software Testing Examination are:

- hold the ISEB Foundation Certificate in Software Testing, AND
- EITHER have at least 18 months experience in software testing,
- OR have completed an ISEB accredited practitioner training course,
- BUT preferably have all three of the above.

The Examination

The examination will consist of:

- one mandatory double-length question (worth 40% of the marks);
- five single-length questions from which three should be answered (worth 20% of the marks per question).

Candidates will be allowed three hours for the examination. The pass mark shall be 60%, and 80% for a distinction.

Objectives of the Practitioner Certificate Qualification

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

The testing skills acquired by attending courses run to this syllabus will be applicable to a wide range of testing challenges that the test practitioner might face. The general approach taught will provide a basis for the formal testing of software systems in general.

The Practitioner Certificate examination will be based on the syllabus in this document. Answers to examination questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable.

Notice to Training Providers

Each major subject heading in the syllabus is assigned an allocated time. The purpose of this is to give both guidance on the relative proportion of time to be allocated to each section of an accredited course and an approximate minimum time for the teaching of each section. Course providers may spend more time than is indicated and candidates may spend more time again in reading and research. The total time specified is 56 hours of lecture and practical work. The course may be delivered as a series of modules with gaps between them, as long as it meets all other constraints. Courses do not have to follow the same order as the syllabus.

Accredited courses will comprise between 30% and 70% practical work overall. This practical work does not have to be split equally across the topics.

There will be a maximum of 16 students per lecturer.

The syllabus contains references to established standards. The use of referenced standards in the preparation of training material is mandatory. Each standard used must be the version quoted in the current version of this syllabus.

Terminology Used

Specialist terminology used in this document is from BS 7925-1:1998; terms not defined in this standard will be as defined in IEEE Std. 829-1998 and IEEE Std. 610-1990.

This Syllabus

The Practitioner Certificate syllabus will be released in two forms; abbreviated and detailed. The abbreviated form will consist of all headings.

Any references to other ISEB syllabuses refer to the latest published version.

The Practitioner Certificate Syllabus

1. INTRODUCTION	4
REVIEW OF THE FOUNDATION CERTIFICATE SYLLABUS	4
TESTING IN THE LIFE CYCLE	4
2. TEST PROCESS.....	4
GENERIC TEST PROCESS	4
TEST PLANNING	5
TEST SPECIFICATION	5
TEST EXECUTION	5
TEST CHECKING AND RECORDING.....	6
CHECKING FOR TEST COMPLETION	6
3. TEST MANAGEMENT	7
TEST MANAGEMENT DOCUMENTATION	7
TEST PLAN DOCUMENTATION	8
TEST ESTIMATION	9
SCHEDULING OF TEST PLANNING	9
TEST PROGRESS MONITORING AND CONTROL.....	9
4. TESTING AND RISK	10
INTRODUCTION TO TESTING AND RISK	10
RISK MANAGEMENT.....	10
5. TEST TECHNIQUES	12
FUNCTIONAL/STRUCTURAL TESTING TECHNIQUES	12
NON-FUNCTIONAL TESTING TECHNIQUES.....	12
DYNAMIC ANALYSIS	12
STATIC ANALYSIS	12
NON-SYSTEMATIC TESTING TECHNIQUES	12
CHOOSING TEST TECHNIQUES	13
6. REVIEWS	13
INTRODUCTION TO REVIEWS	13
THE PRINCIPLES OF REVIEWS	13
INFORMAL REVIEW	13
WALKTHROUGH	14
TECHNICAL REVIEW.....	14
INSPECTION	14
7. INCIDENT MANAGEMENT	15
8. TEST PROCESS IMPROVEMENT.....	15
9. TEST TOOLS	15
OVERVIEW	15
TOOL SELECTION	17
TOOL IMPLEMENTATION.....	17
10. PEOPLE SKILLS.....	18
INDIVIDUAL SKILLS	18
TEST TEAM DYNAMICS	18
FITTING TESTING WITHIN AN ORGANISATION.....	18
MOTIVATION	18

1. Introduction	1½ hours
1.1	Review of the 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.

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).

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

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.

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.

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.

Explain the differences between testing, retesting and regression testing.

2. Test Process	3½ hours
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

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

2.2	Test Planning
------------	----------------------

This topic is covered in detail in section 3.

2.3	Test Specification
------------	---------------------------

Explain that the choice of test specification techniques should be dependent on the risks to be mitigated, the knowledge of the software under test and the source documents, and that the choice should be justified in the test plan/strategy.

Describe 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.

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.

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.

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.

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

Describe the non-functional attributes that may be 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.

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).

2.4	Test Execution
2.4.1	<i>Preparation for Test Execution</i>

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.

2.4.2	<i>Executing the Tests</i>
--------------	-----------------------------------

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.

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.

2.5	Test Checking and Recording
2.5.1	<i>Test Checking</i>

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

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.

2.5.2	<i>Test Recording</i>
--------------	------------------------------

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.

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.

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.

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

3. <i>Test Management</i>	6½ hours
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.

3.1.1 <i>Test Policy</i>

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 in 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.")

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

3.1.2 <i>Test Strategy</i>

Explain that the test strategy 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.

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.

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.

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

3.1.3	<i>Project Test Plan</i>
--------------	---------------------------------

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 identify test cycles based on the software project release plan.
- To satisfy management and users (customers) that adequate testing will be performed for this project.
- To define and communicate the contributions of everybody who is expected to assist in delivering the testing for this project.
- To identify the project deliverable(s) to be tested.

3.1.4	<i>Phase Test Plan</i>
--------------	-------------------------------

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.

3.2	Test Plan Documentation
------------	--------------------------------

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

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.

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.

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.

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.

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.

4. Testing and Risk		4 hours
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.

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

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.

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

4.2.2	Risk Identification	
--------------	----------------------------	--

Introduce the following techniques that are used to identify risks: expert interviews; independent assessment; risk templates; lessons learned; risk workshops; brainstorming and checklists.

Give examples of the appropriate use of the above techniques.

4.2.3	Risk Analysis	
--------------	----------------------	--

Introduce risk analysis as the study of identified risks.

Explain that risk can be calculated as (frequency x 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.

Emphasise that unless trusted risk metrics are available, the analysis will be based on *perceived* 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.

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.

Give examples of both qualitative and a quantitative risk assessments.

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 re-assessed. 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	<i>Risk Mitigation</i>
--------------	-------------------------------

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 preventive action to avoid or reduce risk, and planning for contingent action. 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 preventive action that reduces risk by identifying faults that are subsequently removed.

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:

- Firstly, 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.

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.

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.

5.	<i>Test Techniques</i>	20 hours
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.

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 Myers' The Art of Software Testing for descriptions of the last three techniques].

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 pointers, 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, possible array boundary violations.

Explain that any faults found by compilers are found by static analysis. Compilers find faults in the 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 use 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.

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.

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, 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.

6.	<i>Reviews</i>	7 hours
6.1	Introduction to Reviews	

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

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.

Introduce IEEE Std. 1028-1997 Standard for Software Reviews.

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.

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.

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. 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.

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.

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.

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.

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.

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.

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 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 the reviewers is considered an important factor.

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, and generally 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.

6.6	Inspection
------------	-------------------

The Inspection method to be explained is to be 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 the consensus of the reviewers.

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.

Explain that each reviewer inspects the product individually and reports back on 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, known 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.

7. Incident Management

1½ hours

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.

8. Test Process Improvement

3 hours

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®.

9. Test Tools

6 hours

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 replay 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.
- Test harnesses and drivers 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.
- Test script generators generate test scripts from test specifications.

- Simulators 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).
- Performance test 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.
- Dynamic analysis tools provide run-time information on the state of executing software. These tools are most commonly used to identify unassigned pointers, check pointer arithmetic, monitor the allocation, use and de-allocation of memory to flag memory leaks and highlight other errors difficult to find 'statically'.
- Debugging tools are used mainly 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).
- Comparison 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 tool 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 to automatically 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 built/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 tool 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).

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.

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.

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:

- Champion – 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;
- Change Agent – plans and manages the implementation of the tool (including the pilot project). This person may also be the Champion;
- Tool Custodian – 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	3 hours
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.

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.

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.