

Foundation Syllabus V2.0 – 25 February 1999

<i>Principles of Testing</i>		
<i>Syllabus Topic</i>	<i>Description</i>	<i>Time</i>
Testing Terminology	The BCS SIGIST Standard Glossary of Testing Terms (British Standard BS 7925-1) will be used	5
<p>There is no generally accepted set of testing definitions used by the world-wide testing community. BS 7925-1 exists as a new source of testing definitions.</p>		
Why Testing is Necessary	define errors, faults, failures and reliability; errors and how they occur; cost of errors; exhaustive testing is impossible; testing and risk; testing and quality; testing and contractual requirements; testing and legal, regulatory or mandatory requirements; how much testing is enough	75
<p>An error is a human action that produces an incorrect result. A fault is a manifestation of an error in software (also known as a defect or bug). A fault, if encountered, may cause a failure, which is a deviation of the software from its expected delivery or service. Reliability is the probability that software will not cause the failure of a system for a specified time under specified conditions.</p> <p>Errors occur because we are not perfect and, even if we were, we are working under constraints such as delivery deadlines.</p> <p>A single failure can cost nothing or a lot (e.g. Venus probe). Software in safety-critical systems can cause death or injury if it fails, so the cost of a failure in such a system may be in human lives.</p> <p>Exhaustive testing would in most cases take an enormous amount of resource and is therefore usually impractical.</p> <p>The amount of testing performed depends on the risks involved. Risk must be used as the basis for allocating the test time that is available and for selecting what to test and where to place emphasis.</p> <p>Testing identifies faults, whose removal increases the software quality by increasing the software's potential reliability. Testing is the measurement of software quality. We measure how closely we have achieved quality by testing the relevant factors such as correctness, reliability, usability, maintainability, reusability, testability, etc.</p> <p>Other factors that may determine the testing performed may be contractual requirements, or legal requirements, normally defined in industry-specific standards, or based on agreed best practice (or more realistically non-negligent practice).</p> <p>It is difficult to determine how much testing is enough.</p>		

Fundamental Test Process	the test process; successful tests detect faults; meaning of completion or exit criteria, coverage criteria	45
<p>The fundamental test process comprises planning, specification, execution, recording and checking for completion. In more detail:</p> <p>Test planning.</p> <p>The test plan should specify how the test strategy and project test plan apply to the software under test. This should include identification of all exceptions to the test strategy and of all software with which the software under test will interact during test execution, such as drivers and stubs.</p> <p>Test specification.</p> <p>Test cases should be designed using the test case design techniques selected in the test planning activity.</p> <p>Test execution.</p> <p>Each test case should be executed.</p> <p>Test recording.</p> <p>The test records for each test case should unambiguously record the identities and versions of the software under test and the test specification. The actual outcome should be recorded. It should be possible to establish that all of the specified testing activities have been carried out by reference to the test records.</p> <p>The actual outcome should be compared against the expected outcome. Any discrepancy found should be logged and analysed in order to establish where its cause lies and the earliest test activity that should be repeated, e.g. in order to remove the fault in the test specification or to verify the removal of the fault in the software.</p> <p>The test coverage levels achieved for those measures specified as test completion criteria should be recorded.</p> <p>Checking for test completion.</p> <p>The test records should be checked against the previously specified test completion criteria. If these criteria are not met, the earliest test activity that must be repeated in order to meet the criteria should be identified and the test process should be restarted from that point.</p> <p>It may be necessary to repeat the Test Specification activity to design further test cases to meet a test coverage target.</p> <p>As the objective of a test should be to detect faults, a 'successful' test is one that does detect a fault. This is counter-intuitive, because faults delay progress: a successful test is one that may cause delay. The successful test reveals a fault which, if found later, may be many times more costly to correct so, in the long run, is a good thing.</p> <p>Completion or exit criteria are used to determine when testing (at any test stage) is complete. These criteria may be defined in terms of cost, time, faults found or coverage criteria. Coverage criteria are defined in terms of items that are exercised by test suites, such as branches, user requirements, most frequently used transactions, etc.</p>		

The Psychology of Testing	Testing to find faults; tester-developer relationship; independence	15
<p>Testing is performed with the primary intent of finding faults in the software, rather than of proving correctness. Testing can therefore be perceived as a destructive process. The mindset required to be a tester is different to that of a developer.</p> <p>There are right and wrong ways of presenting faults to authors or management (give examples). It is important to communicate between developer and tester: e.g., changes to the application or menu structures that might affect the tests; or where the developer thinks the code might be buggy; or where there might be difficulty in reproducing reported bugs.</p> <p>Generally it is believed that objective, independent testing is more effective. If author tests then assumptions made are carried into testing, people see what they want to see, there can be emotional attachment, and there may be a vested interest in not finding faults.</p> <p>Levels of independence, such as:</p> <ul style="list-style-type: none"> a) test cases are designed by the person(s) who writes the software under test; b) test cases are designed by another person(s); c) test cases are designed by a person(s) from a different section; d) test cases are designed by a person(s) from a different organisation; e) test cases are not chosen by a person. 		
Re-Testing and Regression Testing	fault-fixing and re-testing; test repeatability; regression testing and automation; selecting regression test cases	30
<p>Whenever a fault is detected and fixed then the software should be re-tested to ensure that the original fault has been successfully removed. You should also consider testing for similar and related faults.</p> <p>Tests should be repeatable, to allow re-testing / regression testing.</p> <p>Regression testing attempts to verify that modifications have not caused unintended adverse side effects in the unchanged software (regression faults) and that the modified system still meets its requirements. It is performed whenever the software, or its environment, is changed.</p> <p>Regression test suites are run many times and generally evolve slowly, so regression testing is ideal for automation. If automation is not possible or the regression test suite is very large then it may be necessary to prune the test suite. You may drop repetitive tests, reduce the number of tests on fixed faults, combine test cases, designate some tests for periodic testing, etc. A subset of the regression test suite may also be used to verify emergency fixes.</p>		
Expected Results	identifying required behaviour	15
<p>Expected results are synonymous with expected outcomes, but not the same as outputs. If expected results have not been defined then a plausible, but erroneous, result may be interpreted as the correct one. Expected results must therefore be defined prior to test execution.</p> <p>The “oracle assumption” is that a tester can routinely identify the correct outcome of a test. An oracle may be (e.g.) the existing system (for a benchmark), or a specification, or an individual’s specialised knowledge, but not the code.</p>		
Prioritisation of Tests	test scope and limited resources; most important tests first; criteria for prioritisation	15
<p>There is never enough time to do all the testing you want, so you must prioritise.</p> <p>Prioritise tests so that whenever you stop testing you have done the best testing in the time available.</p> <p>Identify the ranking criteria used to prioritise, such as severity, probability, visibility of failure, the priorities of the requirements to be tested, what the customer wants, change proneness, error-proneness, business criticality, technical criticality and complexity.</p>		

Testing throughout the lifecycle

Models for Testing	V, V and T; V-model	30
<p>Definitions of verification, validation and testing – as per BS7925-1.</p> <p>The V-model of testing, showing that it identifies baselines (both testing and development deliverables) which should be tested at each stage of development (i.e., testing throughout the life cycle).</p>		
Economics of Testing	early test design; how preparing tests find defects in specifications; cost of faults versus the cost of testing	30
<p>The cost of faults escalates as we move the product towards field use. If a fault is detected before field use, the cost of rework to correct the fault increases dramatically because more than one previous stage of design, coding and testing may have to be repeated. If the fault occurs during field use, the potential cost of the fault might be catastrophic. If faults present in documentation go un-detected, then development based on that documentation might generate many related faults which multiply the effect of the original one.</p> <p>Early test design can prevent fault multiplication. Analysis of specifications during test preparation often brings faults in specifications to light.</p> <p>The cost of testing is generally lower than the cost associated with major faults (such as poor quality product and/or fixing faults), although few organisations have figures to confirm this.</p>		
High Level Test Planning	Scoping the test; risk analysis; test stages; entry and exit criteria; test environment requirements; sources of test data; documentation requirements	30
<p>What to consider, based on IEEE 829-1998 Test Plan Outline.</p> <ol style="list-style-type: none"> 1. Test plan identifier; 2. Introduction; 3. Test items; 4. Features to be tested; 5. Features not to be tested; 6. Approach; 7. Item pass/fail criteria; 8. Suspension criteria and resumption requirements; 9. Test deliverables; 10. Testing tasks; 11. Environmental needs; 12. Responsibilities; 13. Staffing and training needs; 14. Schedule; 15. Risks and contingencies; 16. Approvals. 		

Acceptance Testing	User acceptance testing, contract acceptance testing, alpha & beta testing	30
<p>Acceptance testing may be the only form of testing conducted by and visible to a customer when applied to a software package.</p> <p>User acceptance testing – the final stage of validation. Customer should perform or be closely involved in this. Customers may choose to do any test they wish, normally based on their usual business processes. A common approach is to set up a model office where systems are tested in an environment as close to field use as is achievable.</p> <p>Contract acceptance testing – A demonstration of the acceptance criteria, which would have been defined in the contract, being met.</p> <p>Alpha & beta testing – In alpha and beta tests, when the software seems stable, people who represent your market use the product in the same way(s) that they would if they bought the finished version and give you their comments. Alpha tests are performed at the developer’s site, while beta tests are performed at the testers’ sites.</p>		
Integration Testing in the Large	testing the integration of systems and packages; testing interfaces to external organisations (e.g. Electronic Data Interchange, Internet)	15
<p>Integration with other (complete) systems.</p> <p>Identification of, and risk associated with, interfaces to these other systems.</p> <p>Incremental/non-incremental approaches to integration.</p>		
Non-Functional System Testing	non-functional requirements; non-functional test types: load, performance and stress; security; usability; storage; volume; installability; documentation; recovery	30
<p>Explain that non-functional requirements are as important as functional requirements.</p> <p>Very briefly cover each of the listed techniques. (Load, performance and stress as defined on Systeme Evolutif web pages; security, usability, storage, volume, installability, documentation and recovery as defined in Myers’ book.)</p>		
Functional System Testing	functional requirements; requirements-based testing; business process-based testing	15
<p>Functional requirement as per IEEE definition, which is “A requirement that specifies a function that a system or system component must perform”.</p> <p>Requirements-based testing – where the user requirements specification and the system requirements specification (as used for contracts) may be used to derive test cases.</p> <p>Business process-based testing – based on expected user profiles (e.g. scenarios, use cases, etc.).</p>		
Integration Testing in the Small	assembling components into sub-systems; sub-systems to systems; stubs and drivers; big-bang, top-down, bottom-up, other strategies	15
<p>Integration testing tests interfaces and interaction of modules/subsystems.</p> <p>Role of stubs and drivers.</p> <p>Incremental strategies, to include: top-down, bottom-up and functional incrementation. Non-incremental approach (“big bang”).</p>		
Component Testing	(also known as Unit, Module, Program Testing); overview of BS 7925-2 Software component testing; component test process	30

Maintenance Testing	problems of maintenance; testing changes; risks of changes and regression testing	15
<p>Testing old code – with poor/missing specifications.</p> <p>Scope of testing with respect to changed code.</p> <p>Impact analysis is difficult – so higher risk when making changes – and difficult to decide how much regression testing to do.</p>		
<i>Dynamic Testing Techniques</i>		
Black and White box testing	functional or black-box testing; structural, white or glass box testing; trend from white box to black box through the lifecycle; techniques and tools	30
<p>Explain terminology: black box/functional and white box/structural/glass box.</p> <p>Describe difference between black and white box techniques.</p> <p>Explain that black box is relevant throughout the life cycle whereas, in general, additional white box is appropriate for sub-system testing (unit, link) but becomes progressively less useful towards system and acceptance testing. System and acceptance testers will tend to focus more on specifications and requirements than on code.</p> <p>Emphasise the use of systematic techniques (and corresponding measures) to provide confidence.</p> <p>Explain that tools increase productivity and quality and are particularly useful for white box testing.</p>		
Black box test techniques	Black box techniques as defined in the BCS standard.	45
<p>List all black box techniques in BS 7925-2.</p> <p>Provide description and example of equivalence partitioning and boundary value analysis, plus one other.</p>		
White box test techniques	White box techniques as defined in the BCS standard.	30
<p>List all white box techniques in BS 7925-2.</p> <p>Provide description and example of statement and of branch/decision testing, as per BS 7925-2.</p>		
Error-Guessing	using experience to postulate errors; using error-guessing to complement test design techniques	15
<p>Error-guessing can detect some faults that systematic techniques can miss. Test cases are derived from experience of where errors have occurred in the past or the tester has an insight as to where errors are likely to occur in the future.</p> <p>Error-guessing should be used as a 'mopping-up' approach or as a supplement to systematic techniques, not as the first choice technique.</p>		

<i>Static Testing</i>		
Reviews and the Test Process	Why, when and what to review?; costs and benefits of reviews	30
<p>Why reviews are known to be cost effective.</p> <p>Any document can be reviewed. For instance, requirement specifications, design specifications, code, test plans, user guides, etc. Ideally review as soon as possible.</p> <p>Costs – on-going review costs of approx. 15% of development budget. The cost of reviews includes activities such as the review process itself, metrics analysis and process improvement.</p> <p>Benefits – include areas such as development productivity improvements, reduced development time-scales, testing cost and time reductions, lifetime cost reductions, reduced fault levels, etc.</p>		
Types of Review	types of review; goals, activities performed, roles and responsibilities, deliverables, pitfalls	60
<p>Explain similarities/differences between walkthroughs, inspections, informal reviews, and technical reviews, where each can be identified by the following attributes:</p> <p>Walkthroughs – scenarios, dry runs, peer group, led by author.</p> <p>Inspections – led by trained moderator (not author), defined roles, includes metrics, formal process based on rules and checklists with entry and exit criteria.</p> <p>Informal reviews – undocumented, but useful, cheap, widely-used.</p> <p>Technical reviews (also known as peer reviews) – documented, defined fault-detection process, includes peers and technical experts, no management participation.</p> <p>Goals – validation and verification against specifications and standards, (and process improvement). Achieve consensus.</p> <p>Activities – planning, overview meeting, preparation, review meeting, and follow-up (or similar).</p> <p>Roles and responsibilities – moderators, authors, reviewers/inspectors and managers (planning activities).</p> <p>Deliverables – product changes, source document changes, and improvements (both review and development).</p> <p>Pitfalls – lack of training, lack of documentation, lack of management support (and failure to improve process).</p>		
Static Analysis	simple static analysis; compiler-generated information; data-flow analysis; control-flow graphing; complexity analysis	30
<p>Explain that static analysis involves no dynamic execution and can detect possible faults such as unreachable code, undeclared variables, parameter type mismatches, uncalled functions and procedures, possible array bound violations, etc.</p> <p>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.</p> <p>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 ‘use’ of a variable after it is ‘killed’.</p> <p>Explain use of, and provide example of production of control flow graph for a program.</p> <p>Introduce complexity metrics, including cyclomatic complexity.</p>		

<i>Test Management</i>		
Organisation	organisational structures for testing; team composition	15
<p>Explain that organisations may have different testing structures: testing may be the developer's responsibility, or may be the team's responsibility (buddy testing), or one person on the 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 does the testing.</p> <p>A multi-disciplinary team with specialist skills is usually needed. Most of the following roles are required: test analysts to prepare strategies and plans, test automation experts, database administrator or designer, user interface experts, test environment management, etc.</p>		
Configuration Management	typical symptoms of poor CM; configuration identification; configuration control; status accounting; configuration auditing.	15
<p>Describe typical symptoms of poor CM such as: unable to match source and object code, unable to identify which version of a compiler generated the object code, unable to identify the source code changes made in a particular version of the software, simultaneous changes are made to the same source code by multiple developers (and changes lost), etc.</p> <p>Configuration identification requires that all configuration items (CI) and their versions in the test system are known.</p> <p>Configuration control is maintenance of the CIs in a library and maintenance of records on how CIs change over time.</p> <p>Status accounting is the function recording and tracking problem reports, change requests, etc.</p> <p>Explain that configuration auditing is the function to check on the contents of libraries, etc. for standards compliance, for instance.</p> <p>CM can be very complicated in environments where mixed hardware and software platforms are being used, but sophisticated cross-platform CM tools are increasingly available.</p>		
Test Estimation, Monitoring and Control	test estimation; test monitoring; test control.	30
<p>Test estimation - explain that the effort required to perform activities specified in the high-level test plan must be calculated in advance and that rework must be planned for.</p> <p>Test monitoring – describe useful measures for tracking progress (e.g. number of tests run, tests passed/failed, incidents raised and fixed, retests, etc.). Explain that the test manager may have to report on deviations from the project/test plans such as running out of time before completion criteria achieved.</p> <p>Test control – explain that the re-allocation of resources may be necessary, such as changes to the test schedule, test environments, number of testers, etc.</p>		
Incident Management	what is an 'incident'; incidents and the test process; incident logging; tracking and analysis.	15
<p>An incident is any significant, unplanned event that occurs during testing that requires subsequent investigation and/or correction. Incidents are raised when expected and actual test results differ. Incidents may be raised against documentation as well as code or a system under test.</p> <p>Incidents may be analysed to monitor the test process and to aid in test process improvement.</p> <p>Incidents should be logged when someone other than the author of the product under test performs the testing. Typically the information logged on an incident will include expected and actual results, test environment, software under test id, name of tester(s), severity, scope, priority and any other information deemed relevant to reproducing and fixing the potential fault.</p> <p>Incidents should be tracked from inception through various stages to eventually close out and resolution.</p>		

Standards for Testing	QA standards; industry-specific standards; testing standards	5
<p>Explain that QA standards simply specify that testing <u>should be</u> performed, while industry-specific standards specify <u>what level</u> of testing to perform, and testing standards specify <u>how to</u> perform testing. Ideally testing standards should be referenced from the other two. Examples are ISO 9000, Railway Signalling standard, BS 7925-1, BS 7925-2.</p>		
<p><i>Tool Support for Testing (CAST)</i></p>		
Types of CAST Tool	requirements testing; static analysis; test design; data preparation; character-based test running; GUI test running; test harnesses, drivers and simulators; performance testing; dynamic analysis; debugging; comparison; test management; coverage measurement	45
<p>Requirements testing tools provide automated support for the verification and validation of requirements models, such as consistency checking and animation.</p> <p>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.</p> <p>Test design tools generate test cases from a specification that must normally be held in a CASE tool repository or from formally specified requirements held in the tools itself. Some tools generate test cases from an analysis of the code.</p> <p>Test 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.</p> <p>Character-based test running tools provide test capture and replay facilities for dumb-terminal based applications. The tools simulate user-entered terminal keystrokes and capture screen responses for later comparison. Test procedures are normally captured in a programmable script language, data, test cases and expected results may be held in separate test repositories. These tools are most often used to automate regression testing.</p> <p>GUI test running tools provide test capture and replay facilities for WIMP interface based applications. The tools simulate mouse movement, button clicks and keyboard inputs 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, data, test cases and expected results may be held in separate test repositories. These tools are most often used to automate regression testing.</p> <p>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. 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).</p> <p>Performance test tools have two main facilities: load generation and test transaction measurement. 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 on the architecture. 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.</p> <p>Dynamic analysis tools provide run-time information on the state of executing software. These tools are most 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'.</p> <p>Debugging tools are used mainly by programmers to reproduce bugs 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.</p> <p>(continued)</p>		

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 may also have workflow-oriented facilities to track and control the allocation, correction and re-testing of incidents. 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 without affecting 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.

Tool Selection and Implementation

which test activities can be automated?; CAST tool requirements; which tool types to use?; test process maturity and 'CAST readiness'; selection process; tools, platforms and CAST integration; pilot projects and roll-out

30

There are many test activities which can be automated and test execution tools are not necessarily the first or only choice. Identify your test activities where tool support could be of benefit and prioritise the areas of most importance.

The fit with your test process may be more important than choosing the tool with the most features in deciding whether you need a tool, and which one you choose. The benefits of tools usually depend on a systematic and disciplined test process. If testing is chaotic, the tools may not be useful and may hinder testing. You must have a good process now, or recognise that your process must improve in parallel with tool implementation. The ease by which CAST tools can be implemented might be called 'CAST readiness'.

Tools may have interesting features, but may not necessarily be available on your platforms. E.g. 'works on 15 flavours of Unix, but not yours...'. Some tools, e.g. performance testing tools, require their own hardware, so the cost of procuring this hardware should be a consideration in your cost benefit analysis. If you already have tools, you may need to consider the level and usefulness of integration with other tools. E.g., you may want a test execution tool to integrate with your existing test management tool (or vice versa). Some vendors offer integrated toolkits, e.g. test execution, test management, performance-testing bundles. The integration between some tools may bring major benefits, in other cases, the level of integration is cosmetic only.

Once automation requirements are agreed, the selection process has 4 stages:

1. Creation of a candidate tool shortlist
2. Arrange demos
3. Evaluation(s) of selected tool(s)
4. Review and select tool.

Before making a commitment to implementing the tool across all projects, a pilot project is usually undertaken to ensure the benefits of using the tool can actually be achieved. The objectives of the pilot are to gain some experience in use of the tools, identify changes in the test process required and assess the actual costs and benefits of implementation. Roll out of the tool should be based on a successful result from the evaluation of the pilot. Roll-out normally requires strong commitment from tool users and new projects, as there is an initial overhead in using any tool in new projects.

[End of document]