ISTQB Foundation Level Flash Cards

Technical Review

A Technical Review (also known as a peer review), is considered to be a formal review type, even though no Managers are expected to attend. It involves a structured encounter, in which a peer/s analyse the work with a view to improve the quality of the original work.

- Ideally led by the Moderator
- Attended by peers / technical experts
- Documentation is required
- No Management presence
- Decision making
- Solving technical problems

Walkthrough Review

A walkthrough is a set of procedures and techniques designed for a peer group, lead by the author to review software code. It is considered to be a fairly informal type of review. The walkthrough takes the form a meeting, normally between one and two hours in length.

- Led by the Author
- Attended by a peer group
- Varying level of formality
- Knowledge gathering
- Defect finding

Inspection Review

An inspection is a formal type of review. It requires preparation on the part the review team members before the inspection meeting takes place. A follow-up stage is also a requirement of the inspection. This ensures that any re-working is carried out correctly.

- Led by a Moderator
- Attended by specified roles
- Metrics are included
- Formal process
- Entry and Exit Criteria
- Defect finding

Informal Review

An informal review is an extremely popular choice early on in the development lifecycle of both software and documentation. The review is commonly performed by peer or someone with relevant experience, and should be informal and brief.

- Low cost
- No formal process
- No documentation required
- Widely used review

V & V

Software Validation and Verification can involve analysis, reviewing, demonstrating or testing of all software developments. This will include the development process and the development product itself. Verification and Validation is normally carried out at the end of the development lifecycle (after all software developing is complete). But it can also be performed much earlier on in the development lifecycle by simply using reviews.

Validation

Validation involves the actual testing. This should take place after verification phase has been completed.

Validation: *confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use have been fulfilled.*

Validation: Are we building the right product?

Verification

Verification would normally involve meetings and reviews and to evaluate the documents, plans, requirements and specifications. This can be achieved by using reviews and meetings etc.

Verification: *confirmation by examination and provision of objective evidence that specified requirements have been fulfilled.*

Verification: Are we building the product right?

Waterfall Model

The Waterfall model is also known as the ‘Sequential model’. Each stage follows on from the previous one. The testing is performed in ‘block’ as the last stage.

Planning or Test creation is not considered until the actual software code has been written. This can result in problems being found much later in the project lifecycle than is desirable.

V - Model

The V-Model is an industry standard framework that shows clearly the software development lifecycle in relation to testing. It also highlights the fact that the testing is just as important as the software development itself. The relationships between development and testing are clearly defined.

The V-Model improves the presence of the testing activities to display a more balanced approach.

Spiral Model

The Spiral model is an incremental testing approach to both Development and Testing. This is used most effectively when the users do not know all of the requirements. From what is known, initial requirements can be defined. Then from these the code and test cases are created. As time goes on, more details of the requirements are known and implemented in further iterations of design, coding and testing phases. The system is considered to be complete, when enough of the iterations have taken place.

RAD

RAD represents Rapid Application Development, and is a software development process that was developed in the mid 1980's. It was developed to overcome the rigidity of such processes as 'The Waterfall Model'.

Elements:

- Prototyping
- Iterative Development
- Time-boxing
- Team Members
- Management Approach
- RAD Tools

What is an Error?

Error: A human action that produces an incorrect result.

An example of an **error** would be the misspelling of a variable within the program code.

What is a fault?

Fault: A manifestation of an error in software. A fault, if encountered may cause a failure.

An example of a **fault** would be a variable that is never used within the program code.

Component Testing

Component testing is also known as Unit, Module, or Program Testing. In simple terms, this type of testing focuses simply on testing of the individual components themselves.

It is common for component testing to be carried out by the Developer of the software. This however has a very low rating of testing independence.

Component Integration Testing

This type of Integration testing is concerned with ensuring the interactions between the software components at the module level behave as expected. Component Integration Testing is also often referred to as 'Integration Testing in the Small'.

It is commonly performed after any Component Testing has completed, and the behaviour tested may cover both functional and non-functional aspects of the integrated system.

Requirements Based Functional Testing

Requirements-based Testing is simply testing the functionality of the software/system based on the requirements. The tests themselves should be derived from the documented requirements and not based on the software code itself. This method of functional testing ensures that the users will be getting what they want, as the requirements document basically specifies what the user has asked for.

Business Process Functional Testing Different types of users may use the developed software in different ways. These ways are analysed and business scenarios are then created. User profiles are often used in Business Process Functional Testing. Remember that all of the functionality should be tested for, not just the most commonly used areas.	**Load Testing** Testing the ability of the system to be able to bear loads. An example would be testing that a system could process a specified amount of transactions within a specified time period. So you are effectively loading the system up to a high level, then ensuring it can still function correctly whilst under this heavy load.	**Performance Testing** A program/system may have requirements to meet certain levels of performance. For a program, this could be the speed of which it can process a given task. For a networking device, it could mean the throughput of network traffic rate. Often, Performance Testing is designed to be negative, i.e. prove that the system does not meet its required level of performance.	**Stress Testing** Stress Testing simply means putting the system under stress. The testing is not normally carried out over a long period, as this would effectively be a form of duration testing. Imagine a system was designed to process a maximum of 1000 transactions in an hour. A stress test would be seeing if the systems could actually cope with that many transactions in a given time period. A useful test in this case would be to see how the system copes when asked to process more than 1000.
Security Testing A major requirement in today's software/systems is security, particularly with the internet revolution. Security testing is focused at finding loopholes in the programs security checks. A common approach is to create test cases based on known problems from a similar program, and test these against the program under test.	**Useability Testing** This is where consideration is taken into account of how the user will use the product. It is common for considerable resources to be spent on defining exactly what the customer requires and simple it is to use the program to achieve there aims. For example; test cases could be created based on the Graphical User Interface, to see how easy it would be to use in relation to a typical customer scenario.	**Storage Testing** This type of testing may focus on the actual memory used by a program or system under certain conditions. Also disk space used by the program/system could also be a factor. These factors may actually come from a requirement, and should be approached from a negative testing point of view.	**Volume Testing** Volume Testing is a form of Systems Testing. It primary focus is to concentrate on testing the systems while subjected to heavy volumes of data. Testing should be approached from a negative point of view to show that the program/system cannot operate correctly when using the volume of data specified in the requirements.

Installability Testing A complicated program may also have a complicated installation process. Consideration should be made as to whether the program will be installed by a customer or an installation engineer. Customer installations commonly use some kind of automated installation program. This would obviously have to under go significant testing in itself, as an incorrect installation procedure could render the target machine/system useless.	**Documentation Testing** Documentation in today's environment can take several forms, as the documentation could be a printed document, an integral help file or even a web page. Depending of the documentation media type, some example areas to focus on could be, spelling, usability, technical accuracy etc.	**Recovery Testing** Recovery Testing is normally carried out by using test cases based on specific requirements. A system may be designed to fail under a given scenario, for example if attacked by a malicious user; the program/system may have been designed to shut down. Recovery testing should focus on how the system handles the failure and how it handles the recovery process.	**System Integration Testing** This type of Integration Testing is concerned with ensuring the interactions between systems behave as expected. It is commonly performed after any Systems Testing has completed. Typically not all systems referenced in the testing are controlled by the developing organization. Some systems maybe controlled by other organizations, but interface directly with the system under test.
UAT User Acceptance Testing or 'UAT' is commonly the last testing performed on the software product before its actual release. It is common for the customer to perform this type of testing, or at least be partially involved. Often, the testing environment used to perform User Acceptance Testing is based on a model of the customer's environment. This is done to try and simulate as closely as possible the way in which the software product will actually be used by the customer.	**Contract & Regulation Acceptance Testing** This type of Acceptance Testing is aimed at ensuring the acceptance criteria within the original contract have indeed been met by the developed software. Normally any acceptance criteria is defined when the contract is agreed. Regulation Acceptance Testing is performed when there exists specific regulations that must be adhered to, for example, there may be safety regulations, or legal regulations.	**Operational Acceptance Testing** This form of acceptance testing is commonly performed by a System Administrator and would normally be concerned with ensuring that functionality such as; backup/restore, maintenance, and security functionality is present and behaves as expected.	**Alpha Testing** Alpha Testing should be performed at the developer's site, and predominantly performed by internal testers only. Often, other company department personnel can act as testers. The marketing or sales departments are often chosen for this purpose.

Beta Testing

Beta Testing is commonly performed at the customer's site, and normally carried out by the customers themselves. Potential customers are often eager to trial a new product or new software version. This allows the customer to see any improvements at first hand and ascertain whether or not it satisfies their requirements. On the flip side, it gives invaluable feedback to the developer, often at little or no cost.

Re-Test

It is imperative that when a fault is fixed it is re-tested to ensure the fault has indeed been correctly fixed.

Re-test:
"Whenever a fault is detected and fixed then the software should be re-tested to ensure that the original fault has been successfully removed."

Regression Testing

When checking a fixed fault, you can also consider checking that other existing functionality has not been adversely affected by the fix. This is called Regression Testing.

Regression Test:
"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."

Test Plan Document

A Test Plan should be a single document that basically contains what is going to be tested, why it is going to be tested, and how it is going to be tested. It is also important to clarify what is not going to be tested in the software product too. With regards to using a standard Test Plan layout, then we can look to the advice given by the IEEE(Institute of Electrical and Electronic Engineers) located in the International Standard IEEE Std 929-1998.

Generic Test Process

A standard test process that is commonly used exists within the BS7925-2 Standard for Software Component Testing:

- Test Planning
- Test Specification
- Test Execution
- Test Checking & Recording
- Checking for Test Completion

Test Policy

This should apply to both new projects and maintenance work. Normally fairly short in length, the test policy should be a high-level document, and should contain the following items:

- Definition of testing
- The testing process
- Evaluation of testing
- Quality levels
- Improvement approach

Test Strategy

Based on the test policy, the test strategy is designed to give an overview of the test requirements for a programme or even organization.

Information relating to risks should be documented here, specifically the risks that will be addressed by the testing, and the specific tests that will be used against each risk.

Project Plan

Exactly how the test strategy for a particular project will be implemented is displayed in the project plan. The project test plan will normally be referenced from the overall project plan. In relation to the test strategy, the project plan should detail items from the test strategy that it is complying with, and also items it is not complying with.

Phase Test Plan The specific details of the approach taken by the testing for a specific test phase is shown in this document. It can be thought of as being based on the project plan, but with greater amounts of detail included, such as testing activities based on day to day plan, or expected amounts of man hours to complete individual tasks.	**What is a Failure?** **Failure:** Deviation of the software from its expected delivery or service. An example of a **failure** would be the fact that the warning message is never displayed when it is required.	**Quality & Testing** Once the faults have been rectified, then we can say that the quality of the software product has increased. The testing term 'Quality' can be thought of as an overall term, as the quality of the software product is dependant upon many factors. In general, quality software should be reasonably bug free, delivered on time and within the original budget.	**Test Planning & Control** Test Planning basically involves determining what is going to be tested, why it is going to be tested, and how it is going to be tested. In order to meet the objectives of the testing, it is very important not only to have good plans, but also to ensure that they are adhered to. Test Control will help to achieve this, as its purpose is to ensure that the ongoing testing progress is compared to the Test Plan.
Test Analysis & Design Test objectives can come from a variety of sources. This will commonly take the form of a requirements list or specification. Once the requirements are clearly understood, it should be possible to design tests or conditions based upon these requirements.	**Equivalence Partitioning** What this method allows you to do is effectively partition the possible program inputs. For each of the above input fields, it should not matter which values are entered as long as they are within the correct range and of the correct type. So the point of equivalence portioning is to reduce the amount of testing by choosing a small selection of the possible values to be tested, as the program will handle them in the same way.	**Boundary Value Analysis** By the use of equivalence partitioning, a tester can perform effective testing without testing every possible value. This method can be enhanced further by another method called 'Boundary Value Analysis'. After time, an experienced Tester will be often realise that problems can occur at the boundaries of the input and output spaces. When testing only a small amount of possible values, the minimum and maximum possible values should be amongst the first items to be tested.	**Classification Tree Method** The classification tree method is also known as a decision tree method and the terms can be used interchangeably as they mean the same thing. A decision tree can be learned by splitting the source set into subsets based on an attribute value test. This process is repeated on each subset in a recursively. The recursion is completed when splitting is either not possible, or a single classification can be applied to each element of the subset.

State Transition Testing This type of Black-box testing is based on the concept of 'states' and 'finite-states', and is based on the tester being able to view the software's states, transition between states, and what will trigger a state change. Test cases can then be designed to execute the state changes.	**Statement Testing** This testing method involves using a model of the source code which identifies statements. These statements are the categorized as being either 'executable' or 'non-executable'. In order to use this method, the input to each component must be identified. Also, each test case must be able to identify each individual statement. Lastly, the expected outcome of each test case must be clearly defined	**Branch Decision Testing** This test method uses a model of the source code which identifies individual decisions, and their outcomes. A 'decision' is defined as being an executable statement containing its own logic. This logic may also have the capability to transfer control to another statement. Each test case is designed to exercise the decision outcomes. In order to use this method, the input to each component must be identified.	**Branch Condition Testing** Branch Condition Testing uses a model of the source code, and identifies decisions based on *individual* Boolean operands within each decision condition. A 'decision' is defined as being an executable statement containing its own logic. An example of a decision would be a 'loop' in a program.
Branch Condition Combination Testing Branch Condition Combination Testing uses a model of the source code, and identifies decisions based on *combinations* of Boolean operands within decision conditions. This logic may also have the capability to transfer control to another statement. The decision condition is a Boolean expression which is evaluated to determine the outcome of the decision.	**Requirements Based Functional Testing** Requirements-based Testing is simply testing the functionality of the software/system based on the requirements. The tests themselves should be derived from the documented requirements and not based on the software code itself. This method of functional testing ensures that the users will be getting what they want, as the requirements document basically specifies what the user has asked for.	**Useability Testing** This is where consideration is taken into account of how the user will use the product. It is common for considerable resources to be spent on defining exactly what the customer requires and how simple it is to use the program to achieve there aims. For example; test cases could be created based on the Graphical User Interface, to see how easy it would be to use in relation to a typical customer scenario.	**Volume Testing** Volume Testing is a form of Systems Testing. It primary focus is to concentrate on testing the systems while subjected to heavy volumes of data. Testing should be approached from a negative point of view to show that the program/system cannot operate correctly when using the volume of data specified in the requirements.

Performance Testing

A program/system may have requirements to meet certain levels of performance. For a program, this could be the speed of which it can process a given task. For a networking device, it could mean the throughput of network traffic rate. Often, Performance Testing is designed to be negative, i.e. prove that the system does not meet its required level of performance.

Stress Testing

Stress Testing simply means putting the system under stress. The testing is not normally carried out over a long period, as this would effectively be a form of duration testing. Imagine a system was designed to process a maximum of 1000 transactions in an hour. A stress test would be seeing if the systems could actually cope with that many transactions in a given time period. A useful test in this case would be to see how the system copes when asked to process more than 1000.

Dynamic Analysis

Dynamic analysis is a testing method that can provide information on the state of software. It can achieve this dynamically i.e. it provides information when the software is actually running. It is commonly used to exercise parts of the program that use memory resources e.g.:

- Memory allocation
- Memory usage
- Memory de-allocation
- Memory leaks
- Unassigned pointers

Static Analysis

Static Analysis is a set of methods designed to analyse software code in an effort to establish it is correct, prior to actually running the software. As we already know, the earlier we find a fault the cheaper it is to fix. So by using Static Analysis, we can effectively test the program even before it has been written. This would obviously only find a limited number of problems, but at least it is something that can be done very early on in the development lifecycle.

Control Flow Graphing

Control flow graphs display the logic structure of software. The flow of logic through the program is charted. It is normally used only by Developers as it is a very low level form testing, often used in Component Testing.

It can be used to determine the number of test cases required to test the programs logic. It can also provide confidence that the detail of the logic in the code has been checked.

Cyclomatic Complexity

Cyclomatic Complexity is a software metric that is used to measure the complexity of a software program. Once we know now how complex the program is, we then know how easy it will be to test.

$$C = E - N + P$$

- C = Cyclomatic Complexity
- E = number of edges
- N = number of nodes
- P = number of components

Lines of Code

The most basic form of a complexity metric is the ‘Lines of Code’ metric, or ‘LOC’ metric. Its purpose like other complexity metrics is to estimate the amount of effort that will be required not only to develop such a program, but also assist in estimating how much effort will be required to test it.

In its simplest form we could use the LOC metric by literally counting the number of lines of code in the program.

Data Flow Analysis

The idea behind Data-flow Analysis is to work-out the dependencies between items of data that are used by a program. When a program is ran, it rarely runs in a sequential order i.e. starting at line 1 and finishing at line 100. What usually happens is that the dependencies of the data within the program will determine the order. Data-flow Analysis can be used to find ‘definitions’ that have no intervening ‘use’. Data-flow analysis is also used to detect variables that are ‘used’ after it has effectively been ‘killed’.

Error Guessing Why can one Tester find more errors than another Tester in the same piece of software? More often than not this is down to a technique called 'Error Guessing'. To be successful at Error Guessing, a certain level of knowledge and experience is required. A Tester can then make an educated guess at where potential problems may arise. This could be based on the Testers experience with a previous iteration of the software, or just a level of knowledge in that area of technology.	**Exploratory Testing** This type of testing is normally governed by time. It consists of using tests based on a test chapter that contains test objectives. It is most effective when there are little or no specifications available. It should only really be used to assist with, or compliment a more formal approach. It can basically ensure that major functionality is working as expected without fully testing it.	**Ad-hoc Testing** This type of testing is considered to be the most informal, and by many it is considered to be the least effective. Ad-hoc testing is simply making up the tests as you go along. Often, it is used when there is only a very small amount of time to test something. A common mistake to make with Ad-hoc testing is not documenting the tests performed and the test results. Even if this information is included, more often than not additional information is not logged such as, software versions, dates, test environment details etc.	**Random Testing** A Tester normally selects test input data from what is termed an 'input domain'. Random Testing is simply when the Tester selects data from the input domain 'randomly'. As you can tell, there is little structure involved in 'Random Testing'. In order to avoid dealing with the above questions, a more structured Black-box Test Design could be implemented instead. However, using a random approach could save valuable time and resources if used in the right circumstances.
Quality Assurance Standards A Quality Assurance (QA) standard simply specifies that testing should be performed. *Example: ISO 9000*	**Industry Specific Standards** An industry specific standard will detail exactly what level of testing is to be performed. *Examples:* ▪ *Railway Signalling standard* ▪ *DO-178B* ▪ *Nuclear Industry standard* ▪ *MISRA guidelines for motor vehicle software* ▪ *Pharmaceutical standards*	**Testing Standards** Testing standards will detail how to perform the testing. Ideally, a testing standard should be referenced from a QA or Industry specific standard. *Example: BS7925-1, BS7925-2*	**Review Definition** Review: A process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users or other interested parties for comment or approval. [IEEE] A review should be performed when all of the supporting documentation is available. This can include design documents, requirements documents, standards documents, basically any documentation that has either been influential or is applicable to the document to be reviewed.

Review Roles

Organisations will commonly have different named roles than those listed below, but this will give you an idea of a commonly used set of roles used throughout the world.

- Manager
- Moderator
- Author
- Reviewer
- Scribe

Review Process Structure

An example of a typical review process is below. This is probably the most documented review process you will find in the software development world, and is open to interpretation:

- Planning
- Kick-off
- Preparation
- Meeting
- Rework
- Follow-up
- Exit Criteria

Incident Management

We term an incident; any significant, unplanned event that occurs during testing that requires subsequent investigation and/or correction. The incident should be raised when the actual result differs from the expected result. After the inevitable investigation of the incident, there may be a reason other than a software fault, for example:

- Test environment incorrectly set up
- Incorrect Test Data used
- Incorrect Test Specification

IEEE Std. 1044-1993

This standard aims to provide a standard approach to classification of anomalies found in software. It includes descriptions of the processes involved in a software life cycle, including details on how anomalies should be recorded and subsequently processed. It consists of four sequential steps; Recognition, Investigation, Action, Disposition. Each of those steps has three administrative activities which are; Recording, Classifying, Identifying Impact.

Test Implementation and Execution

This stage is where the actual testing is performed. It can mean running the test cases manually or by use of an automated testing tool. Before this can happen though, the Test Case designs need to be made into actual Test Cases.

Evaluating Exit Criteria and Reporting

This stage is designed to ensure that any specified Exit Criteria has been met by the performed testing activities. The Exit Criteria should have been previously specified in the Test Planning stage. The stored Test Results can be checked in this stage against the Exit Criteria. If the Exit Criteria has not been met, then more tests may be required, or even changes to the Exit Criteria may be recommended. This is a good stage to create a Test Summary.

Test Closure Activities

This stage is concerned with collecting test results and test related documentation in order to achieve a milestone prior to a release of the product. Any defects found during the testing should have been fixed and verified fixed at this stage. As with most development projects there are always problems encountered, here is a good stage to evaluate those and attempt to learn any lessons for future developments.

Developer Attributes

A Developer's attributes:

Highly valued within the company
Industry standard qualifications
Seen as being creative
Poor communicators
Skills in a very specific area

Tester Attributes A Tester's attributes: Rarely valued within a company No industry standard qualifications Seen as being destructive Very good communicators Multi-talented	**Reviews - Manager** Manager The Manager will be the person who makes the decision to hold the review. Managing people's time with respect to the review is also a Managers responsibility.	**Reviews - Moderator** Moderator The Moderator effectively has overall control and responsibility of the review. They will schedule the review, control the review, and ensure any actions from the review are carried out successfully. Training may be required in order to carry out the role of Moderator successfully.	**Reviews - Author** Author The Author is the person who has created the item to be reviewed. The Author may also be asked questions in the review.
Reviews - Reviewer Reviewer The reviewers are the attendees of the review who are attempting to find errors in the item under review. They should come from different perspectives in order to provide a well balanced review of the item.	**Reviews - Scribe** Scribe The Scribe (or Recorder) is the person who is responsible for documenting issues raised during the process of the review meeting.	**Formal Review Process** The review process: Planning Kick-off Preparation Meeting Rework Follow-up Exit Criteria	**Static Analysis Tools** By examining the code instead of running test cases through the code, this type of tool can provide information on the actual quality of the software. Cyclomatic complexity is one such characteristic that can be obtained by using this type of tool.

Test Design Tools This type of tool can generate test cases from specifications, which are normally stored in a CASE tool repository. Some variations of this type of tool can also generate test cases from analysing the code itself.	**Test Data Preparation Tool** Data can be selected from existing test specific databases by using this type of tool. Advanced types of this tool can utilise a range of database and file formats.	**Test Execution Tools** These are an extremely popular type of tool. They provide capture and replay facilities for WIMP interface based applications. The tools can simulate mouse movement, mouse clicks and keyboard inputs. The tools can even recognize windows and buttons, thus making them extremely versatile. The test procedures are normally written in a specific scripting language. This tool is another popular choice for regression testing.	**Test Harnesses** If the software under test does not have a user interface, then test harnesses and drivers can be used to execute the software. These types of tools can be bought off the shelf, but more commonly they are built for a specific purpose.
Incident Management Tools This type of tool stores and manages any incident reports. Prioritisation of incident reports can be achieved by this tool. Automatic assigning of actions, and status reporting is also a common feature of this type of tool.	**Performance Test Tools** This type of tool comprises of two components; Load Generation and Test Transaction Measurement., Load Generation is commonly performed by running the application using its interface or by using drivers. The number of transactions performed this way are then logged. Performance test tools will commonly be able to display reports and graphs of load against response time.	**Dynamic Analysis Tools** Run-time information on the state of the executing software is achieved by using Dynamic Analysis Tools. These tools are ideally suited for monitoring the use and allocation of memory. Faults such as memory leaks, unassigned pointers can be found, which would otherwise be difficult to find manually.	**Review Process Support Tools** This type of tool provides features such as storing review comments, review processes, traceability between documents and source code.

Test Comparators

This type of tool is used to highlight differences between actual results and expected results. Off the shelf Comparison Tools can normally deal with a range of file and database formats. This type of tool often has filter capabilities to allow 'ignoring' of rows or columns of data or even areas on a screen

Test Management Tools

Test Management Tools commonly have multiple features. Test Management is mainly concerned with the management, creation and control of test documentation. More advanced tools have additional capabilities such as test management features, for example; result logging and test scheduling.

Coverage Measurement Tools

This type of tool provides objective measures of structural test coverage when the actual tests are executed. Before the programs are compiled, they are first instrumented. Once this has been completed they can then be tested. The instrumentation process allows the coverage data to be logged whilst the program is running. Once testing is complete, the logs can provide statistics on the details of the tests covered.

Modelling Tools

Several different types of modelling tools exist today, they can range from finding defects in state models, object models and data models. Valuable defects can be found using modelling tools, with the added benefit of finding them early in the development lifecycle.

Monitoring Tools

These tools are typically used for testing e-commerce and e-business applications. The main purpose of this tool is to check web sites to ensure that they are available to customers and also to produce warnings if problems are detected.

Security Testing Tools

These tools are commonly used for testing e-commerce and e-business applications, and sometimes web sites. A security testing tool will check for any parts of a web based system that could cause potential security risks if attacked.

Requirements Management Tools

This type of tool is designed to assist with verification and validation of requirements, for example; consistency checking. They are also designed to store requirements statements, which assists with traceability.

Tool Selection Process

A suggested tool selection and evaluation process is:

- Determine the actual problem or requirement
- Ensure that there are no obvious alternative solutions
- Prepare a business case
- Identify any constraints
- Identify any specific required tool features or characteristics
- Prepare a short-list of possible suitable tools
- Perform a detailed evaluation
- Perform a competitive trial, if needed

Pilot Projects The last thing we want is to introduce a tool into the organisation, only to find a few weeks down the line it fails resulting in potentially disastrous scenarios. In order to avoid this situation, we can implement a pilot project. The benefits of using a pilot project are; ▪ Gaining experience using the tool ▪ Identify any test process changes ▪ Identify any shortcomings suitability of the tool	**Review - Planning** There should be awareness of any company policies, product requirements and/or project plans that may provide specific requirements in order for a review to take place. Definition of Entry & Exit criteria, personnel selection are also done at this stage.	**Review – Kick-off** Normally a week before the planned review, the Moderator ensures that the item is ready for review and distributed. The Moderator also briefs the attendees on their roles and responsibilities.	**Review - Preparation** All review participants examine the item to be reviewed. If standards are used, then checks for deviation form the standards should be used. Comparison to similar items can also be used.
Review - Meeting The review normally lasts between 1 – 2 hours. All items on the agenda/checklist are worked through. Findings are noted by the recorder. Comments are aimed at the review item not the author.	**Review - Rework** The Moderator and Scribe document the findings in a Review Summary and report it to the Manager. The Review Summary will contain defects found and actions of follow-up work to be carried out.	**Review - Follow-up** The Moderator ensures that all additional work by the author is checked for completeness and correctness. An additional review may be required; dependant on the amount/complexity of re-work undertaken.	**Review – Exit Criteria** Exit Criteria can take the form of ensuring that all actions are completed, or that any uncorrected items are properly documented, possibly in a defect tracking system.

The Test Leader The Test Leader will commonly come from a testing background and have a full understanding of how testing is performed. They will also possess good managerial expertise. They are also responsible for ensuring that test coverage is sufficient and will be required to produce reports.	**The Tester** The Tester obviously provides the skills necessary to perform the Testing itself. This role can include test design and test execution. Automated testing skills are also a possible requirement of this role. The following list shows some example activities you might expect a Tester to perform: ▪ Create Test Specifications ▪ Preparation of test data ▪ Execute tests and provide results	**The Client** The client is effectively the project sponsor, and will provide the budget for the project. The Client can also be the business owner.	**The Project Manager** Management skills are provided by the Project Manager. The Project Manager will be actively involved throughout the project and will provide feedback to the client.
The Developer A Developer will provide the skills to write the actual software code and perform Unit Testing. They may also be called upon at a later stage to provide bug fixes and technical advice.	**The Business Analyst** The Business Analyst will provide knowledge of the business and analysis skills. The Business Analyst will also be responsible for creating User Requirements based on talks with the Users.	**The Systems Analyst** Systems design will be provided by the Systems Analyst. The Systems Analyst will also be responsible for developing the Functional Specification from the User Requirements.	**The Technical Designer** Technical detail and support to the system design is the responsibility of the Technical Designer. This role may include database administration.

ISTQB Self Learning Materials

www.istqb.guru