Q. 1: How to differentiate between work-products and products?

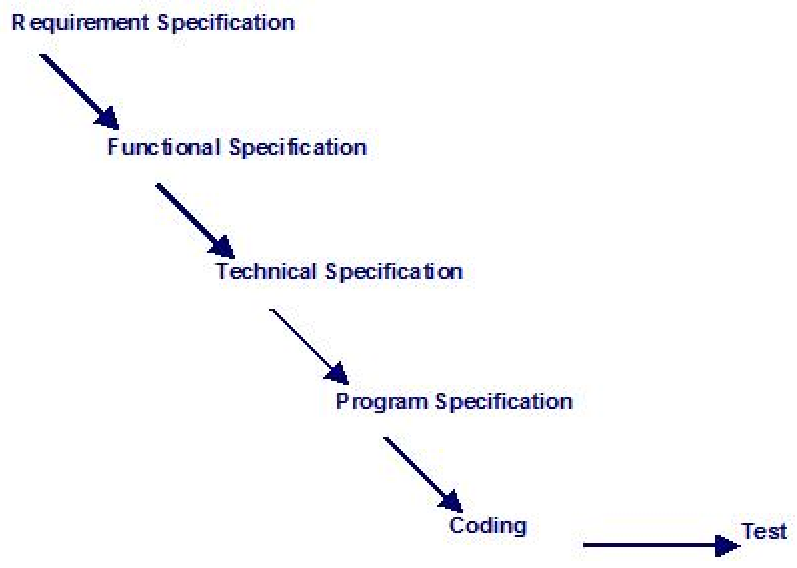
A work-product is an intermediate deliverable required to create the final product. Work-products can be documentation or code. The code and associated documentation will become the product when the system is declared ready for release. In software development, work-products are generally created in a series of defined stages, from capturing a customer requirement, to creating the system, to delivering the system. These stages are usually shown as steps within a software development life cycle.

<<<<<< =================== >>>>>>

Q. 2: What is the simple waterfall development model.

A development life cycle for a software product involves capturing the initial requirements from the customer, expanding on these to provide the detail required for code production, writing the code and testing the product, ready for release.

A simple development model is shown below. This is known traditionally as the waterfall model.



The waterfall model shown above shows the steps in sequence where the customer requirements are progressively refined to the point where coding can take place. This type of model is often referred to as a linear or sequential model. Each work-product or activity is completed before moving on to the next.

In the waterfall model, testing is carried out once the code has been fully developed. Once this is completed, a decision can be made on whether the product can be released into the live environment.

This model for development shows how a fully tested product can be created, but it has a significant drawback: what happens if the product fails the tests? Let us look at a simple case study.

<<<<<< =================== >>>>>>

Q. 3: At what point of time do we need to perform quality check during software development?

In the waterfall model, the testing at the end serves as a quality check. The product can be accepted or rejected at this point.

Unlike in a mechanical manufacturing industry, in software development, it is not possible to simply reject the parts of the system found to be defective, and release the rest. The nature of software functionality is such that removal of software is often not a clean-cut activity - this action could well cause other areas to function incorrectly. It may even cause the system to become unusable.

In addition, we may not be able to choose not to deliver anything at all. The commercial and financial effects of this course of action could be substantial.

What is needed is a process that assures quality throughout the development life cycle. At every stage, a check should be made that the work-product for that stage meets its objectives. This is a key point, work-product evaluation taking place at the point where the product has been declared complete by its creator. If the work-product passes its evaluation (test), we can progress to the next stage in confidence. In addition, finding problems at the point of creation should make fixing any problems cheaper than fixing them at a later stage. This is the cost escalation model.

The checks throughout the life cycle include verification and validation.

<<<<<< =================== >>>>>>

Q. 4: How do we distinctly apply verification and validation to work-products during software development?

Verification - checks that the work-product meets the requirements set out for it. An example of this would be to ensure that a website being built follows the guidelines for making websites usable by as many people as possible. Verification helps to ensure that we are building the product in the right way.

Validation - changes the focus of work-product evaluation to evaluation against user needs. This means ensuring that the behavior of the work-product matches the customer needs as defined for the project. For example, for the same website above, the guidelines may have been written with people familiar with websites in mind. It may be that this website is also intended for novice users. Validation would include these users checking that they too can use the website easily. Validation helps to ensure that we are building the right product as far as the users are concerned.

Testing helps to ensure that the work-products are being developed in the right way (verification) and that the product will meet the user needs (validation).

<<<==================== >>>>>><<<

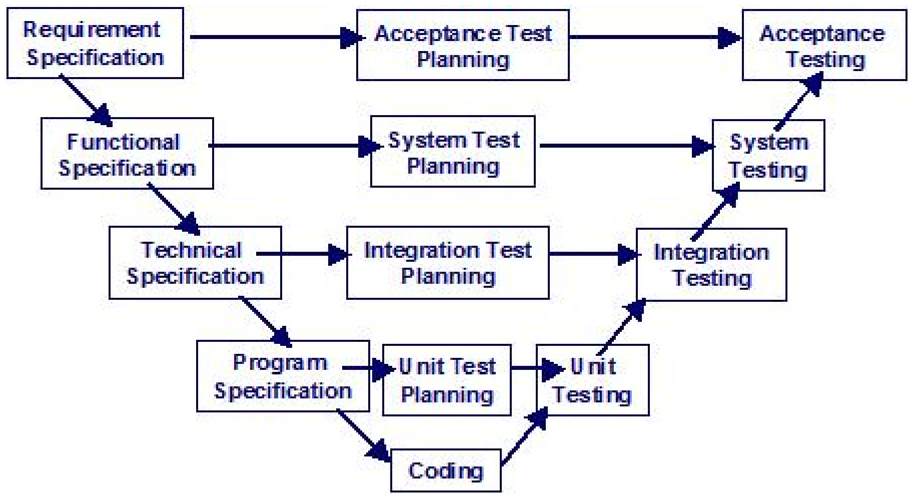
Q. 5: How many types of software development models do we have that facilitate early work-product evaluation?

There are two types of development models that facilitate early work-product evaluation.

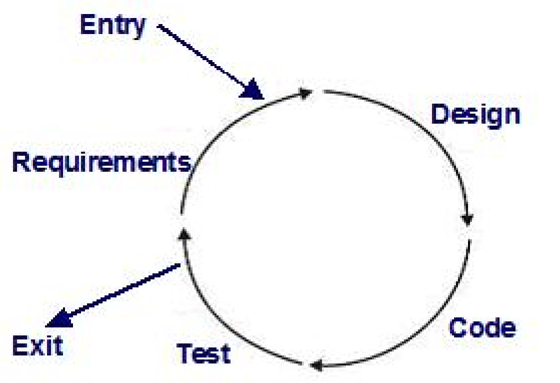
The first is an extension to the waterfall model, known as the V-model.

The second is a cyclical model, where the coding stage often begins once the initial user needs have been captured. Cyclical models are often referred to as iterative models.

1) V-Model (Sequential Development Model)



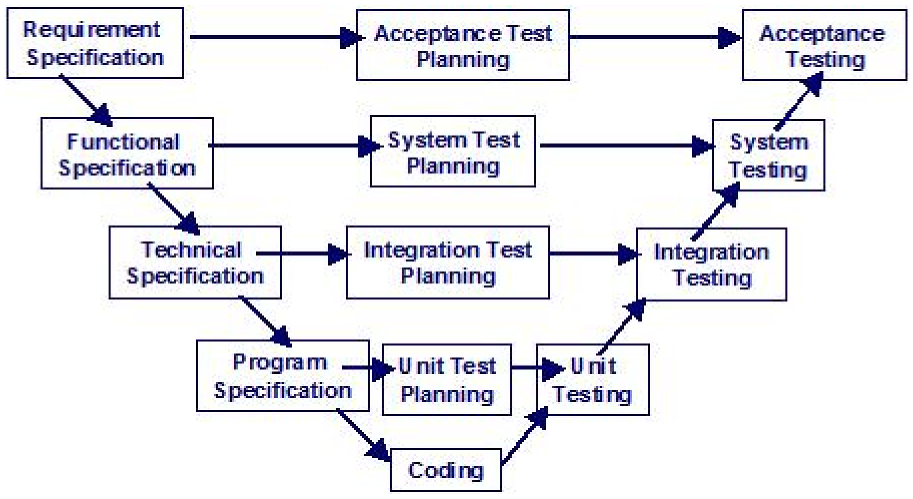
2) Iterative-Incremental Development Model



Iterative-Incremental Development Model

Q. 6: How would you explain the different branches of a typical V-model for software development?

Refer following figure explaining a typical V-model



A) Left Side Branch: As for the waterfall model, the left-hand side of the model focuses on elaborating the initial requirements, providing successively more technical detail as the development progresses. In the model shown, these are:

1) Requirement specification - capturing of user needs.

2) Functional specification - definition of functions required to meet user needs.

3) Technical specification - technical design of functions identified in the functional specification.

4) Program specification - detailed design of each module or unit to be built to meet required functionality.

These specifications can be reviewed to check for the following:

a) Conformance to the previous work-product (so in the case of the functional specification, verification would include a check against the requirement specification).

b) That there is sufficient detail for the subsequent work-product to be built correctly (again, for the functional specification, this would include a check that there is sufficient information in order to create the technical specification).

c) That it is testable - is the detail provided sufficient for testing the work-product?

B) Middle Branch: The middle of the V-model shows that planning for testing should start with each work-product. Thus, using the requirement specification as an example, acceptance testing would be planned for, right at the start of the development.

C) Right Side Branch: The right-hand side focuses on the testing activities. For each work-product, a testing activity is identified. These are shown in figure above

1) Testing against the requirement specification takes place at the acceptance testing stage.

2) Testing against the functional specification takes place at the system testing stage.

3) Testing against the technical specification takes place at the integration testing stage.

4) Testing against the program specification takes place at the unit testing stage.

This allows testing to be concentrated on the detail provided in each work-product, so that defect can be identified as early as possible in the life cycle, when the work-product has been created.

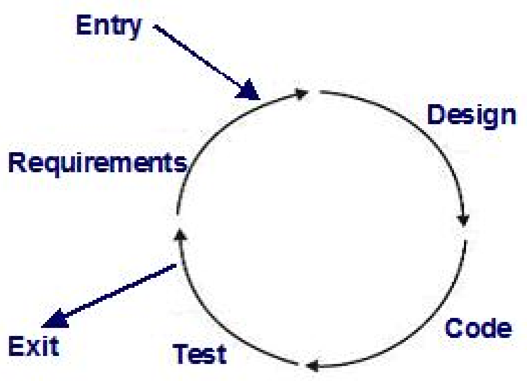
Remembering that each stage must be completed before the next one can be started, this approach to software development pushes validation of the system by the user representatives right to the end of the life cycle. If the customer needs were not captured accurately in the requirement specification, or if they change, then these issues may not be uncovered until the user testing is carried out. Fixing problems at this stage could be very costly; in addition, it is possible that the project could be cancelled altogether.

<<<<<< =================== >>>>>>

Q. 7: How would you explain the iterative development model for software development?

This is one where the requirements do not need to be fully defined before coding can start. Instead, a working version of the product is built, in a series of stages, or iterations - hence the name iterative or incremental development. Each stage encompasses requirements definition, design, code and test.

Refer following figure explaining typical Iterative Development Model



Iterative-Incremental Development Model

An Iterative model for development has fewer steps, but involves user from the beginning. These steps are typically:

1) Define Iteration Requirements
2) Buid Iteration
3) Test Iteration

This type of development is often referred to as cyclical - we go 'round the development cycle a number of times', within the project. The project will have a defined time-scale and cost. Within this, the cycles will be defined. Each cycle will also have a defined time-scale and cost. The cycles are commonly referred to as time-boxes. For each time-box, a requirement is defined and a version of the code is produced, which will allow testing by the user representatives. At the end of each time-box, a decision is made on what extra functionality needs to be created for the next iteration. This process is then repeated until a fully working system has been produced.

A key feature of this type of development is the involvement of user representatives in the testing. Having the users represented throughout minimizes the risk of developing an unsatisfactory product. The user representatives are empowered to request changes to the software, to meet their needs.

This approach to software development can pose problems, however.

The lack of formal documentation makes it difficult to test. To counter this, developers may use test-driven development. This is where functional tests are written first, and code is then created and tested. It is reworked until it passes the tests.

In addition, the working environment may be such that developers make any changes required, without formally recording them. This approach could mean that changes cannot be traced back to the requirements or to the parts of the software that have changed. Thus, traceability as the project progresses is reduced. To mitigate this, a robust process must be put in place at the start of the project to manage these changes (often part of a configuration management process

Another issue associated with changes is the amount of testing required to ensure that implementation of the changes does not cause unintended changes to other parts of the software (this is called regression testing.

Forms of iterative development include prototyping, rapid application development (RAD) and agile software development. A proprietary methodology is the Rational Unified Process (RUP).

<<<<<< =================== >>>>>>

Q. 8: What are the significant benefits of an iterative model for software development?

1) The involvement of user representatives in the testing.

2) Minimized risk of developing an unsatisfactory product due to the availability of users representative throughout.

3) The user representatives are empowered to request changes to the software, to meet their needs.

<<<<<< =================== >>>>>>

Q. 9: What are the attributes of good testing across the development life cycle?

Some of the characteristics of good testing effort across the development life cycle are:

1) Early test design: In the V-model, the test planning begins with the specification documents. This activity is part of the fundamental test process. After test planning, the documents would be analyzed and test cases designed. This approach would ensure that testing starts with the development of the requirements, i.e. a proactive approach to testing is undertaken. As we saw in iterative development, test-driven development may be adopted, pushing testing to the front of the development activity.

2) Each work-product is tested: In the V-model, each document on the left is tested by an activity on the right. Each specification document is called the test basis, i.e. it is the basis on which tests are created. In iterative development, each release is tested before moving on to the next.

3) Testers are involved in reviewing requirements before they are released: In the V-model, testers would be invited to review all documents from a testing perspective.

<<<<<< =================== >>>>>>

Q. 10: What are the different levels of testing?

The typical levels of testing are:

1) Unit (Component) testing
2) Integration testing
3) System testing
4) Acceptance testing

Each of these test levels will include tests designed to uncover problems specifically at that stage of development. These levels of testing can be applied to iterative development also. In addition, the levels may change depending on the system. For instance, if the system includes some software developed by external parties, or bought off the shelf, acceptance testing on these may be conducted before testing the system as a whole.

Q. 11: What is Unit or Component Testing?

Before testing of the code can start, clearly the code has to be written. This is shown at the bottom of the V-model. Generally, the code is written in component parts, or units. The units are usually constructed in isolation, for integration at a later stage. Units are also called programs, modules or components.

Unit testing is intended to ensure that the code written for the unit meets its specification, prior to its integration with other units.

In addition to checking conformance to the program specification, unit testing would also verify that all of the code that has been written for the unit can be executed. Instead of using the specification to decide on inputs and expected outputs, the developer would use the code that has been written for this.

The test bases for unit testing can include:

1) The component requirements
2) The detailed design
3) The code itself

Unit testing requires access to the code being tested. Thus test objects (i.e. what is under test) can be the components, the programs, data conversion/migration programs and database modules. Unit testing is often supported by a unit test framework. In addition, debugging tools are often used.

An approach to unit testing is called Test Driven Development. As its name suggests, test cases are written first, code built, tested and changed until the unit passes its tests. This is an iterative approach to unit testing.

Unit testing is usually performed by the developer who wrote the code (and who may also have written the program specification). Defects found and fixed during unit testing are often not recorded.

<<<<<< =================== >>>>>>

Q. 12: What is Integration Testing?

Once the units have been written, the next stage would be to put them together to create the system. This is called integration. It involves building something large from a number of smaller pieces.

The purpose of integration testing is to expose defects in the interfaces and in the interactions between integrated components or systems.

The test bases for integration testing can include:

1) The software and system design
2) A diagram of the system architecture

3) Workflows and use-cases

The test objects would essentially be the interface code. This can include subsystems' database implementations.

Before integration testing can be planned, an integration strategy is required. This involves making decisions on how the system will be put together prior to testing.

There are three commonly quoted integration strategies, namely:

1) Big-Bang Integration
2) Top-Down Integration
3) Bottom-Up Integration

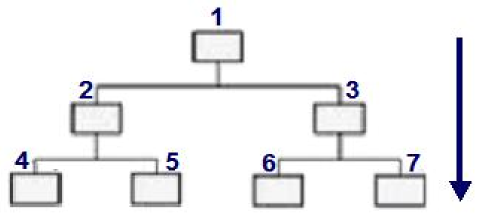
<<<<<< =================== >>>>>>

Q. 13: How would you explain the different Integration Testing Strategies?

1) Big-Bang Integration: This is where all units are linked at once, resulting in a complete system. When testing of this system is conducted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.

This type of integration is generally regarded as a poor choice of integration strategy. It introduces the risk that problems may be discovered late in the project, where they are more expensive to fix.

2) Top-Down Integration: This is where the system is built in stages, starting with components that call other components. Components that call others are usually placed above those that are called. Top-down integration testing will permit the tester to evaluate component interfaces, starting with those at the ‘top’ as shown in the following figure.



Top-Down Integration

The control structure of a program can be represented in a chart. In the above figure, component 1 can call components 2 and 3. Thus in the structure, component 1 is placed above components 2 and 3. Component 2 can call components 4 and 5. Component 3 can call components 6 and 7. Thus in the structure, components 2 and 3 are placed above components 4 and 5 and components 6 and 7, respectively.

In this chart, the order of integration might be:

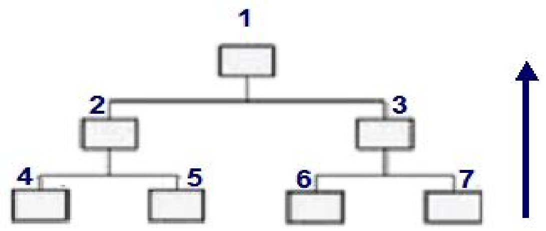
1,2 or 1,3 or 2,4 or 2,5 or 3,6 or 3,7

Top-down integration testing requires that the interactions of each component must be tested when it is built. Those lower down in the hierarchy may not have been built or integrated yet. In the above figure, in order to test component 1's interaction with component 2, it may be necessary to replace component 2 with a substitute since component 2 may not have been integrated yet.

This is done by creating a skeletal implementation of the component, called a stub. A stub is a passive component, called by other components. In this example, stubs may be used to replace components 4 and 5, when testing component 2.

The use of stubs is commonplace in top-down integration, replacing components not yet integrated.

3) Bottom-up Integration: This is the opposite of top-down integration and the components are integrated in a bottom-up order as shown in following figure.



Bottom-Up Integration

The integration order might be:

4,2 or 5,2 or 6,3 or 7,3 or 2,1 or 3,1

Hence, in bottom-up integration, components 4-7 would be integrated before components 2 and 3. In this case, the components that may not be in place are those that actively calls other components. As in top-down integration testing, they must be replaced by specially written components. When these special components call other components, they are called drivers.

They are so called because, in the functioning program, they are active, controlling other components.

Components 2 and 3 could be replaced by drivers when testing components 4 - 7. They are generally more complex than stubs.

There may be more than one level of integration testing. For example:

Component integration testing focuses on the interactions between software components and is done after component (unit) testing. Developers usually carry out this type of integration testing.

System integration testing focuses on the interactions between different systems and may be done after system testing of each individual system. For example, a trading system in an investment bank will interact with the stock exchange to get the latest prices for its stocks and shares on the international market. Testers usually carry out this type of integration testing.

It should be noted that testing at system integration level carries extra elements of risk. These can include: at a technical level, cross-platform issues; at an operational level, business workflow issues; and at a business level, risks associated with ownership of regression issues associated with change in one system possibly having a knock-on effect on other systems.

<<<<<< =================== >>>>>>

Q. 14: What is System Testing?

Having checked that the components all work together at unit level, the next step is to consider the functionality from an end-to-end perspective. This activity is called system testing.

System testing is necessary because many of the criteria for test selection at unit and integration testing result in the production of a set of test cases that are unrepresentative of the operating conditions in the live environment. Thus testing at these levels is unlikely to reveal errors due to interactions across the whole system, or those due to environmental issues.

System testing serves to correct this imbalance by focusing on the behavior of the whole system/product as defined by the scope of a development project or program, in a representative live environment. A team that is independent of the development process usually carries it out. The benefit of this independence is that an objective assessment of the system can be made, based on the specifications as written, and not the code.

In the V-model, the behavior required of the system is documented in the functional specification. It defines what must be built to meet the requirements of the system. The functional specification should contain definitions of both the functional and non-functional requirements of the system.

A functional requirement is a requirement that specifies a function that a system or system component must perform. Functional requirements can be specific to a system. For instance, you would expect to be able to search for flights on a travel agent's website, whereas you would visit your online bank to check that you have sufficient funds to pay for the flight.

Thus functional requirements provide detail on what the application being developed will do.

Non-functional system testing looks at those aspects that are important but not directly related to what functions the system performs. These tend to be generic requirements, which can be applied to many different systems. In the example above, you can expect that both systems will respond to your inputs in a reasonable time frame, for instance. Typically, these requirements will consider both normal operations and behavior under exceptional circumstances.

The amount of testing required at system testing, however, can be influenced by the amount of testing carried out (if any) at the previous stages. In addition, the amount of testing advisable would also depend on the amount of verification carried out on the requirements.

Test bases for system testing can include:

1) System and software requirement specifications
2) Use cases
3) Functional specifications
4) Risk analysis reports

5) System, user and operation manuals

The test object will generally be the system under test.

<<<<<< =================== >>>>>>

Q. 15: Describe some examples of non-functional requirements that details the performance of the application

Some of the non-functional requirements are:

1) Install ability: Installation procedures.

2) Maintainability: Ability to introduce changes to the system.

3) Performance: Expected normal behavior.

4) Load handling: Behavior of the system under increasing load.

5) Stress handling: Behavior at the upper limits of system capability.

6) Portability: Use on different operating platforms.

7) Recovery: Recovery procedures on failure.

8) Reliability: Ability of the software to perform its required functions over time.

9) Usability: Ease with which users can engage with the system

Q. 16: What is Acceptance Testing?

The step after the system testing is usually acceptance testing. The purpose of acceptance testing is to provide the end users with confidence that the system will function according to their expectations. Referring once more to the V-model, acceptance testing will be carried out using the requirement specification as a basis for test.

The requirement specification is typically the first document to be written, after initial capture of the user requirement. An example of a requirement could be to create a website that enables users to buy airline tickets online.

The subsequent documentation (functional, technical and program specifications) will expand on this in increasing levels of detail, in order to facilitate development of the system, as seen earlier. Thus, it is paramount that these requirements are fully documented and correct before further development activity is carried out. Again, this is the V-model approach. We know that having such an ideal set of requirements is a rare thing. This does not mean, however, that the need for correctness and completeness should be ignored.

As for system testing, no reference is made to the code from which the system is constructed.

The test bases for acceptance testing can include:

1) User requirements
2) System requirements
3) Use cases
4) Business processes
5) Risk analysis reports

The test objects for system testing can include:

1) The fully integrated system
2) Forms and reports produced by the system.

Unlike system testing, however, the testing conducted here should be independent of any other testing carried out. Its key purpose is to demonstrate system conformance to, for example, the customer requirements and operational and maintenance processes. For instance, acceptance testing may assess the system's readiness for deployment and use.

Acceptance testing is often the responsibility of the customers or users of a system, although other project team members may be involved as well.

<<<<<< =================== >>>>>>

Q. 17: What are the different forms of Acceptance Testing?

Typical forms of acceptance testing include the following:

1) User acceptance testing: Testing by user representatives to check that the system meets their business needs. This can include factory acceptance testing, where the system is tested by the users before moving it to their own site. Site acceptance testing could then be performed by the users at their own site.

2) Operational acceptance testing: Often called operational readiness testing. This involves checking that the processes and procedures are in place to allow the system to be used and maintained. This can include checking:

# Back Facilities
# Procedure for disaster recovery
# Training for end users
# Maintenance procedures
# Data load and migration tasks
# Security procedures

3) Contract and Regulation Acceptance Testing: Operational acceptance testing: Often

# Contract acceptance testing: Sometimes the criteria for accepting a system are documented in a contract. Testing is then conducted to check that these criteria have been met, before the system is accepted.

# Regulation acceptance testing: in some industries, systems must meet governmental, legal or safety standards. Examples of these are the defence, banking and pharmaceutical industries.

4) Alpha and Beta Testing:

# Alpha testing takes place at the developer's site - the operational system is tested whilst still at the developer's site by internal staff, before release to external customers. Note that testing here is still independent of the development team.

# Beta testing takes place at the customer's site - the operational system is tested by a group of customers, who use the product at their own locations and provide feedback, before the system is released. This is often called 'field testing'.

<<<<<< =================== >>>>>>

Q. 18: What are the different types or categories of Testing?

Types of testing fall into the following categories:

1) Functional Testing: The functional testing looks at the specific functionality of a system, such as searching for flights on a website, or perhaps calculating employee pay correctly using a payroll system. Note that security testing is a functional test type. Another type of functional testing is interoperability testing - this evaluates the capability of the system to interact with other specified components.

Functional testing is also called specification-based testing: testing against a specification.

2) Non-Functional Testing: This is where the behavioral aspects of the system are tested. As for functional testing, these requirements are usually documented in a functional specification. Thus, mainly black-box testing techniques are used for this type of testing.

These tests can be referenced against a quality model, such as the one defined in ISO 9126 Software Engineering - Software Product Quality. Note that a detailed understanding of this standard is not required for the exam.

3) Structural Testing: This type of testing is used to measure how much testing has been carried out. In functional testing, this could be the number of functional requirements tested against the total number of requirements.

In structural testing, we change our measure to focus on the structural aspects of the system. This could be the code itself, or an architectural definition of the system. We want to do this to check the thoroughness of the testing carried out on the system that has actually been built. A common measure is to look at how much of the actual code that has been written has been tested.

The structural testing can be carried out at any test level.

<<<<<< =================== >>>>>>

Q. 19: Describe different type of testing related to changes

We carry out testing at the different stages in the development life cycle. At any level of testing, it can be expected that defects will be discovered. When these are found and fixed, the quality of the system being delivered can be improved.

After a defect is detected and fixed the changed software should be retested to confirm that the problem has been successfully removed. This is called retesting or confirmation testing. Note that when the developer removes the defect, this activity is called debugging, which is not a testing activity. Testing finds a defect, debugging fixes it.

The unchanged software should also be retested to ensure that no additional defects have been introduced as a result of changes to the software. This is called regression testing. Regression testing should also be carried out if the environment has changed.

Regression testing involves the creation of a set of tests, which serve to demonstrate that the system works as expected. These would be run again many times over a testing project, when changes are made, as discussed above. This repetition of tests makes regression testing suitable for automation in many cases.

<<<<<< =================== >>>>>>

Q. 20: Describe Maintenance Testing

For many projects the system is eventually released into the live environment. Hopefully, once deployed, it will be in service as long as intended, perhaps for years or decades.

During this deployment, it may become necessary to change the system. Changes may be due to the following reasons:

1) Additional features being required.

2) The system being migrated to a new operating platform.

3) The system being retired - data may need to be migrated or archived.

4) Planned upgrade to COTS -based systems.

5) New faults being found requiring fixing (these can be ‘hot fixes’).

Once changes have been made to the system, they will need to be tested (retesting), and it also will be necessary to conduct regression testing to ensure that the rest of the system has not been adversely affected by the changes. Testing that takes place on a system which is in operation in the live environment is called maintenance testing.

When changes are made to migrate from one platform to another, the system should also be tested in its new environment. When migration includes data being transferred in from another application, then conversion testing also becomes necessary.

All changes must be tested, and, ideally, all of the system should be subject to regression testing. In practice, this may not be feasible or cost-effective. An understanding of the parts of the system that could be affected by the changes could reduce the amount of regression testing required. Working this out is termed impact analysis, i.e. analyzing the impact of the changes on the system.

Impact analysis can be difficult for a system that has already been released. This is because the specifications may be out of date (or non-existent), and/or the original development team may have moved on to other projects, or left the organization altogether.

Q. 21: What is the use of static testing techniques?

Static testing techniques are those techniques that test software without executing the code. This includes both the testing of work-products other than code, typically requirements or specification documents, and the testing of code without actually executing it.

There are two types of static testing techniques.

1) Review: It is typically used to find and remove errors and ambiguities in documents before they are used in the development process, thus reducing one source of defects in the code.

Reviews are normally completed manually;

2) Static analysis: It enables code to be analyzed for structural defects or systematic programming weaknesses that may lead to defects.

Static analysis is normally completed automatically using tools.

<<<<<< =================== >>>>>>

Q. 22: What do we do in reviews – one of static testing techniques?

A review is a systematic examination of a document by one or more people with the main aim of finding and removing errors. Giving a draft document to a colleague to read is the simplest example of a review, and one which can usually yield a larger crop of errors than we would have anticipated.

Reviews can be used to test anything that is written or typed; this can include documents such as requirement specifications, system designs, code, test plans and test cases.

Reviews represent the first form of testing that can take place during a software development life cycle, since the documents reviewed are normally ready long before the code has been written.

<<<<<< =================== >>>>>>

Q. 23: What are the key benefits of Reviews?

The practice of testing specification documents by reviewing them early on in the life cycle helps to identify defects before they become part of the executable code, and so makes those defects cheaper and easier to remove. The same defect, if found during dynamic test execution, would incur the extra cost of initially creating and testing the defective code, diagnosing the source of the defect, correcting the problem and rewriting the code to eliminate the defect.

Reviewing code against development standards can also prevent defects from appearing in test execution, though in this case, as the code has already been written, not all the additional costs and delays are avoided.

Important as cost and time saving are, though, there are also other important benefits of finding defects early in the life cycle, among them the following:

1) Development productivity can be improved and time-scales reduced because the correction of defects in early work-products will help to ensure that those work-products are clear and unambiguous. This should enable a developer to move more quickly through the process of writing code. Also, if defects are removed before they become executable code there will be fewer errors to find and fix during test execution.

2) Testing costs and time can be reduced by removing the main delays in test execution, which arise when defects are found after they have become failures and the tester has to wait for a fix to be delivered. By reviewing the code and removing defects before they become failures the tester can move more swiftly through test execution.

3) Reductions in lifetime costs can be achieved because fewer defects in the final software ensure that ongoing support costs will be lower.

4) Improved communication results as authors and their peers discuss and refine any ambiguous content discovered during review to ensure that all involved understand exactly what is being delivered.

<<<<<< =================== >>>>>>

Q. 24: What types of defects are generally found by Reviews?

The types of defects most typically found by reviews are:

1) Deviations from standards either internally defined and managed or regulatory / legally defined by Parliament or perhaps a trade organization.

2) Requirements defects - for example, the requirements are ambiguous, or there are missing elements.

3) Design defects - for example, the design does not match the requirements.

4) Insufficient maintainability - for example, the code is too complex to maintain.

5) Incorrect interface specifications - for example, the interface specification does not match the design or the receiving or sending interface.

6) All reviews aim to find defects, but some types of review find certain types of defects more effectively and efficiently than others.

<<<<<< =================== >>>>>>

Q. 25: What are the different types of review processes?

Review processes can vary widely in their level of formality, where formality relates to the level of structure and documentation associated with the activity. Some types of review are completely informal, while others are very formal.
The decision on the appropriate level of formality for a review is usually based on combinations of the following factors:

1) The maturity of the development process: The more mature the process is, the more formal reviews tend to be.

2) Legal or regulatory requirements: These are used to govern the software development activities in certain industries, notably in safety-critical areas such as railway signaling, and determine what kinds of review should take place.

3) The need for an audit trail: Formal review processes ensure that it is possible to trace backwards throughout the software development life cycle. The level of formality in the types of review used can help to raise the level of audit trail.

Q. 26: What are the different objectives of reviews?

Reviews have different objectives, where the term ‘review objective’ identifies the main focus for a review. Typical review objectives are:

1) Finding defects.
2) Gaining understanding.
3) Generating discussion.
4) Decision making by consensus.

The way a review is conducted will depend on what its specific objective is, so a review aimed primarily at finding defects will be quite different from one that is aimed at gaining understanding of a document.

<<<<<< =================== >>>>>>

Q. 27: What are the different elements of Basic Review Process?

All reviews, formal and informal alike, exhibit the same basic elements of process that are:

1) The reviewers study the document under review.

2) Reviewers identify issues or problems and inform the author either verbally or in a documented form, which might be as formal as raising a defect report or as informal as annotating the document under review.

3) The author decides on any action to take in response to the comments and updates the document accordingly.

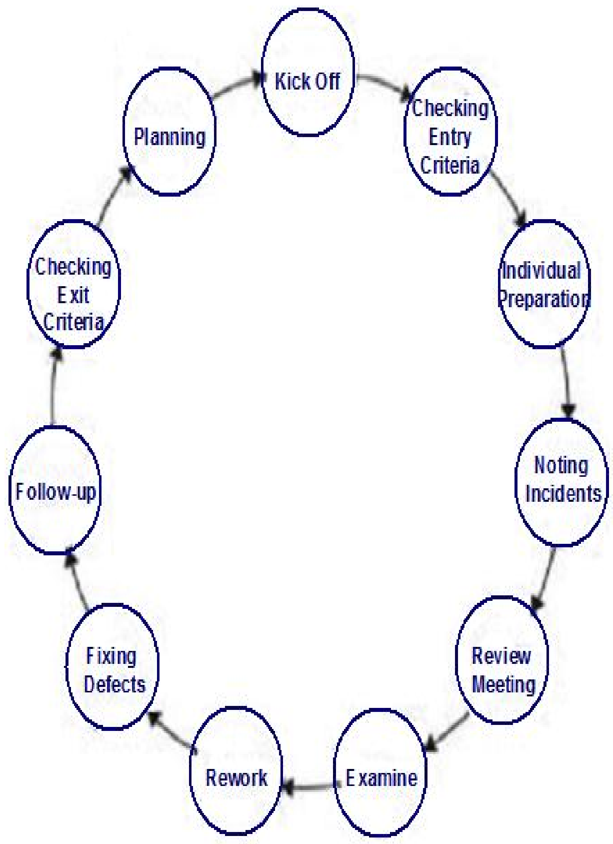
This basic process is always present, but in the more formal reviews it is elaborated to include additional stages and more attention to documentation and measurement.

<<<<<< =================== >>>>>>

Q. 28: What are the different activities of a Formal Review

Reviews at the more formal end of the spectrum, such as technical reviews and inspections, share certain characteristics that differentiate them from the less formal reviews, of which walkthroughs are a typical example.

Key stages in formal reviews are as shown in the following Figure.



Stages of Formal Reviews

1) Planning:

# Selecting the personnel: Ensuring that those selected can and will add value to the process. There is little point in selecting a reviewer who will agree with everything written by the author without question. As a rule of thumb it is best to include some reviewers who are from a different part of the organization, who are known to be ‘picky’, and known to be dissenters.

Reviews, like weddings, are enhanced by including ‘something old, something new, something borrowed, something blue’. In this case ‘something old’ would be an experienced practitioner; ‘something new’ would be a new or inexperienced team member; ‘something borrowed’ would be someone from a different team; ‘something blue’ would be the dissenter who is hard to please. At the earliest stage of the process a review leader must be identified. This is the person who will coordinate all of the review activity.

# Allocating roles: Each reviewer is given a role to provide them with a unique focus on the document under review. Someone in a tester role might be checking for testability and clarity of definition, while someone in a user role might look for simplicity and a clear relationship to business values. This approach ensures that, although all reviewers are working on the same document, each individual is looking at it from a different perspective.

# Defining the entry and exit criteria: Especially for the most formal review types (e.g. inspection).

# Selecting the parts of documents to be reviewed (not always required; this will depend on the size of the document: A large document may need to be split into smaller parts and each part

reviewed by a different person to ensure the whole document is reviewed fully).

2) Kick-off: Distributing documents; explaining the objectives, process and documents to the participants; and checking entry criteria (for more formal review types such as inspections). This can be run as a meeting or simply by sending out the details to the reviewers. The method used will depend on time-scales and the volume of information to pass on. A lot of information can be disseminated better by a meeting rather than expecting reviewers to read pages of text.

3) Review entry criteria: This stage is where the entry criteria agreed earlier are checked to ensure that they have been met, so that the review can continue - this is mainly used in the more formal review types such as inspections.

4) Individual preparation: Work done by each of the participants on their own before the review meeting, which would include reading the source documents, noting potential defects, questions and comments. This is a key task and may actually be time-boxed, e.g. participants may be given two hours to complete the preparation.

5) Noting incidents: In this stage the potential defects, questions and comments found during individual preparation are logged.

6) Review meeting: This may include discussion regarding any defects found, or simply just a log of defects found. The more formal review types like inspections will have documented results or minutes. The meeting participants may simply note defects for the author to correct; they might also make recommendations for handling or correcting the defects. The approach taken will have been decided at the kick-off stage so that all participants are aware of what is required of them.

The decision as to which approach to take may be based on one or all of the following factors:

# Time available (if time is short the meeting may only collect defects).

# Requirements of the author (if the author would like help in correcting defects).

# Type of review (in an inspection only the collection of defects is allowed - there is never any discussion).

7) Examine: This includes the recording of the physical meetings or tracking any group electronic communications.

8) Rework: After a review meeting the author will have a series of defects to correct; correcting the defects is called rework.

9) Fixing defects: Here the author will be fixing defects that were found and agreed as requiring a fix.

10) Follow-up: The review leader will check that the agreed defects have been addressed and will gather metrics such as how much time was spent on the review and how many defects were found. The review leader will also check the exit criteria (for more formal review types such as inspections) to ensure that they have been met.

11) Checking exit criteria: At this stage the exit criteria defined at the start of the process are checked to ensure that all exit criteria have been met so that the review can be officially closed as finished.

<<<<<< =================== >>>>>>

Q. 29: What are the general Roles and Responsibilities of different members in a review team?

The role of each reviewer is to look at documents belonging to them from their assigned perspective; this may include the use of checklists. For example, a checklist based on a particular perspective (such as user, maintainer, tester or operations) may be used, or a more general checklist (such as typical requirements problems) may be used to identify defects.

In addition to these assigned review roles the review process itself defines specific roles and responsibilities that should be fulfilled for formal reviews. They are:

1) Manager: The manager decides on what is to be reviewed (if not already defined), ensures there is sufficient time allocated in the project plan for all of the required review activities, and determines if the review objectives have been met. Managers do not normally get involved in the actual review process unless they can add real value, e.g. they have technical knowledge key to the review.

2) Moderator: The moderator is sometimes known as the review leader. This is the person who leads the review of the document or set of documents, including planning the review, running the meeting, and follow-ups after the meeting. If necessary, the moderator may mediate between the various points of view and is often the person upon whom the success of the review rests. The moderator will also make the final decision as to whether to release an updated document.

3) Author: The author is the writer or person with chief responsibility for the development of the document(s) to be reviewed. The author will in most instances also take responsibility for fixing any agreed defects.

4) Reviewers: These are individuals with a specific technical or business background (also called checkers) who, after the necessary preparation, identify and describe findings (e.g. defects) in the product under review. The reviewers should be chosen to represent different perspectives and roles in the review process and take part in any review meetings.

5) Scribe (or recorder): The scribe attends the review meeting and documents all of the issues and defects, problems and open points that were identified during the meeting.

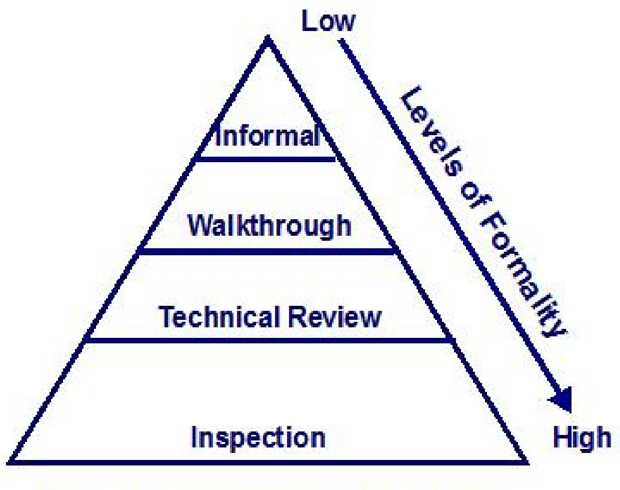
An additional role not normally associated with reviews is that of the tester. Testers have a particular role to play in relation to document reviews. In their test analysis role they will be required to analyze a document to enable the development of tests. In analyzing the document they will also review it, e.g. in starting to build end-to-end scenarios they will notice if there is a 'hole' in the requirements that will stop the business functioning, such as a process that is missing or some data that is not available at a given point. So effectively a tester can either be formally invited to review a document or may do so by default in carrying out the tester's normal test analysis role.

<<<<<< =================== >>>>>>

Q. 30: What is the spectrum of formality of review types?

A single document may be subject to many different review types: for example, an informal review may be carried out before the document is subjected to a technical review or, depending on the level of risk, a technical review or inspection may take place before a walkthrough with a customer.

Following figure shows the different levels of formality by review type.



Levels of Formality of Reviews

Each type of review has its own defining characteristics. There are four types of reviews to cover the spectrum of formality. These are generally known as:

1) Informal review (least formal): Key characteristics are as under

a) There is no formal process underpinning the review.

b) The review may be documented but this is not required; many informal reviews are not documented.

c) There may be some variations in the usefulness of the review depending on the reviewer, e.g. the reviewer does not have the technical skills but is just available to check quickly and ensure that the document makes sense.

d) The main purpose is to find defects and this is an inexpensive way to achieve some limited benefit.

e) The review may be implemented by pair programming (where one programmer reviews the code of the other ‘pair programmer’) or by a technical lead reviewing designs and code.

2) Walkthrough: Key characteristics are as under

a) The meeting is led by the author of the document under review and attended by members of the author's peer group.

b) Review sessions are open-ended and may vary in practice from quite informal to very formal.

c) Preparation by reviewers before the walkthrough meeting, production of a review report or a list of findings, and appointment of a scribe who is not the author are all optional components that are sometimes present.

d) The main purposes are to enable learning about the content of the document under review, to help team members gain an understanding of the content of the document, and to find defects.

e) Walkthroughs typically explore scenarios, or conduct dry runs of code or process.

3) Technical review: Key characteristics are as under

a) Technical reviews are documented and use a well-defined defect detection process that includes peers and technical experts.

b) The review is usually performed as a peer review without management participation and is ideally led by a trained moderator who is not the author.

c) Reviewers prepare for the review meeting, optionally using checklists, and prepare a review report with a list of findings.

d) Technical reviews may vary in practice from the quite informal to very formal and have a number of purposes, including: discussion, decision making, evaluation of alternatives, finding defects, solving technical problems and checking conformance to specifications and standards.

4) Inspection (most formal): Key characteristics are as under

a) Inspections are led by a trained moderator who is not the author and usually involve peer examination of a document; individual inspectors work within defined roles.

b) The inspection process is formal, based on rules and checklists, and uses entry and exit criteria.

c) Pre-meeting preparation is essential, which would include reading of any source documents to ensure consistency.

d) An inspection report, with a list of findings, is produced, which includes metrics that can be used to aid improvements to the process as well as correcting defects in the document under review.

e) After the meeting a formal follow-up process is used to ensure that corrective action is completed and timely.

f) The main purpose is to find defects, and process improvement may be a secondary purpose.

g) In reality the lines between the review types often get blurred and what is seen as a technical review in one company may be seen as an inspection in another. The above is the ‘classic view’ of reviews. The key for each company is to agree the objectives and benefits of the reviews that they plan to carry out.

Q. 31: What are the factors on which the success of reviews mainly depend?

When measuring the success of a particular review the following suggested success factors should be considered:

1) Each review should have a clearly predefined and agreed objective and the right people should be involved to ensure the objective is met. For example, in an inspection each reviewer will have a defined role and therefore needs the experience to fulfil that role; this should include testers as

valued reviewers.

2) Any defects found are welcomed, and expressed objectively.

3) The review should be seen as being conducted within an atmosphere of trust, so that the outcome will not be used for the evaluation of the participants, and that the people issues and psychological aspects are dealt with (e.g. making it a positive experience for the author and all participants).

4) Review techniques (both formal and informal) that are suitable to the type and level of software work-products and reviewers (this is especially important in inspections).

5) Checklists or roles should be used, where appropriate, to increase effectiveness of defect identification; for example, in an inspection, roles such as data entry clerk or technical architect may be required to review a particular document.

6) Management support is essential for a good review process (e.g. by incorporating adequate time for review activities in project schedules).

7) There should be an emphasis on learning and process improvement.

Other more quantitative approaches to success measurement could also be used:

a) How many defects found.

b) Time taken to review/inspect.

c) Percentage of project budget used/saved.

<<<<<< =================== >>>>>>

Q. 32: What do we do in Static Analysis – one of static testing techniques?

Like reviews, static analysis looks for defects without executing the code. However, unlike reviews static analysis is carried out once the code has been written. Its objective is to find defects in software source code and software models.

Source code is any series of statements written in some human-readable computer programming language that can then be converted to equivalent computer executable code - the developer normally generates it.

A software model is an image of the final solution developed using techniques such as Unified Modeling Language (UML); a software designer normally generates it.

Static analysis can find defects that are hard to find during test execution by analyzing the program code, e.g. instructions to the computer can be in the form of control flow graphs (how control passes between modules) and data flows (ensuring data is identified and correctly used).

<<<<<< =================== >>>>>>

Q. 33: What are the benefits of using static analysis – static testing technique?

1) Early detection of defects prior to test execution: As with reviews, the earlier the defect is found, the cheaper and easier it is to fix.

2) Early warning about suspicious aspects of the code or design: By the calculation of metrics, such as a high-complexity measure. If code is too complex it can be more prone to error or less dependent on the focus given to the code by developers. If they understand that the code has to be complex then they are more likely to check and double check that it is correct; however, if it is unexpectedly complex there is a higher chance that there will be a defect in it.

3) Identification of defects not easily found by dynamic testing: Such as development standard breaches as well as detecting dependencies and inconsistencies in software models, such as links or interfaces that were either incorrect or unknown before static analysis was carried out.

4) Improved maintainability of code and design: By carrying out static analysis, defects will be removed that would otherwise have increased the amount of maintenance required after 'go live'. It can also recognize complex code that if corrected will make the code more understandable and therefore easier to maintain.

5) Prevention of defects: By identifying the defect early in the life cycle it is a lot easier to identify why it was there in the first place (root cause analysis) than during test execution, thus providing information on possible process improvement that could be made to prevent the same defect appearing again.

Static analysis tools add the greatest value when used during component and integration testing. This will normally involve their use by developers to check against predefined rules or development standards and by designers during software modelling.

<<<<<< =================== >>>>>>

Q. 34: What type of defects are generally discovered by static analysis tools?

1) Referencing a variable with an undefined value, e.g. using a variable as part of a calculation before the variable has been given a value.

2) Inconsistent interface between modules and components, e.g. module X requests three values from module Y, which has only two outputs.

3) Variables that are never used. This is not strictly an error, but if a programmer declares a variable in a program and does not use it, there is a chance that some intended part of the program has inadvertently been omitted,

4) Unreachable (dead) code. This means lines of code that cannot be executed because the logic of the program does not provide any path in which that code is included.

5) Programming standards violations, e.g. if the standard is to add comments only at the end of the piece of code, but there are notes throughout the code, this would be a violation of standards.

6) Security vulnerabilities, e.g. password structures that are not secure.

7) Syntax violations of code and software models, e.g. incorrect use of the programming or modelling language.

<<<<<< =================== >>>>>>

Q. 35: What are the drawbacks of using static analysis tools?

A static analysis tool runs automatically and reports all defects it identifies, some of which may be insignificant and require little or no work to correct, whilst others could be critical and need urgent correction.

These defects therefore require strong management to ensure that the full benefit is obtained from using the tool in the first place.

Q. 36: What are the main steps involved in the design of tests?

The design of tests comprises of three main steps:

1) Identify test conditions: Decide on a test condition, which would typically be a small section of the specification for our software under test.

Going by the definition - A test condition is some characteristic of our software that we can check with a test or a set of tests.

2) Specify test cases: Design a test case that will verify the test condition.

Going by the definition - A test case gets the system to some starting point for the test (execution preconditions); then applies a set of input values that should achieve a given outcome (expected result), and leaves the system at some end point (execution postcondition).

3) Specify test procedures: Write a test procedure to execute the test, i.e. get it into the right starting state, input the values, and check the outcome.

Going by the definition - A test procedure identifies all the necessary actions in sequence to execute a test. Test procedure specifications are often called test scripts (or sometimes manual test scripts to distinguish them from the automated scripts that control test execution tools.

This is a simple set of steps. Of course we will have to carry out a very large number of these simple steps to test a whole system, but the basic process is still the same. To test a whole system we write a test execution schedule, which puts all the individual test procedures in the right sequence and sets up the system so that they can be run.

The test development process may be implemented in more or less formal ways. In some situations it may be appropriate to produce very little documentation and in others a very formal and documented process may be appropriate. It all depends on the context of the testing, taking account of factors such as maturity of development and test processes, the amount of time available and the nature of the system under test. Safety-critical systems, for example, will normally require a formal test process.

<<<<<< =================== >>>>>>

Q. 37: What is Test Coverage & where do we apply it?

Test coverage provides a quantitative assessment of the extent and quality of testing. It answers the question ‘how much testing have you done?’ in a way that is not open to interpretation.

Vague statements like ‘I am nearly finished’, or ‘I have done two weeks' testing’ or ‘I have done everything in the test plan' generate more questions than they answer. They are statements about how much testing has been done or how much effort has been applied to testing, rather than statements about how effective the testing has been or what has been achieved.

We need to know about test coverage for two very important reasons:

1) It provides a quantitative measure of the quality of the testing that has been done by measuring what has been achieved.

2) It provides a way of estimating how much more testing needs to be done. Using quantitative measures we can set targets for test coverage and measure progress against them.

Statements like 'I have tested 75 per cent of the decisions' or 'I have tested 80 per cent of the requirements' provide useful information. They are neither subjective nor qualitative; they provide a real measure of what has actually been tested. If we apply coverage measures to testing based on priorities, which are themselves based on the risks addressed by individual tests, we will have a reliable, objective and quantified framework for testing.

Test coverage can be applied to any systematic technique; & here it will be applied to specification-based techniques to measure how much of the functionality has been tested, and to structure-based techniques to measure how much of the code has been tested. Coverage measures may be part of the completion criteria defined in the test plan and used to determine when to stop testing.

<<<<<< =================== >>>>>>

Q. 38: What are the different categories of test case design techniques?

The test case design techniques are grouped into three categories like:

1) Those based on deriving test cases directly from a specification or a model of a system or proposed system, known as specification-based or black-box techniques. So black-box techniques are based on an analysis of the test basis documentation, including both functional and non-functional aspects. They do not use any information regarding the internal structure of the component or system under test.

2) Those based on deriving test cases directly from the structure of a component or system, known as structure-based, structural or white-box techniques. We will concentrate on tests based on the code written to implement a component or system, but other aspects of structure, such as a menu structure, can be tested in a similar way.

3) Those based on deriving test cases from the tester's experience of similar systems and general experience of testing, known as experience-based techniques.

<<<<<< =================== >>>>>>

Q. 39: What are the specification-based techniques for test case design?

It was originally called 'black-box' because this technique take a view of the system that does not need to know what is going on 'inside the box'.

The specification-based techniques derive test cases directly from the specification or from some other kind of model of what the system should do. The source of information on which to base testing is known as the ‘test basis’. If a test basis is well defined and adequately structured we can easily identify test conditions from which test cases can be derived.

The most important point about specification-based techniques is that specifications or models do not (and should not) define how a system should achieve the specified behavior when it is built; it is a specification of the required (or at least desired) behavior. One of the hard lessons that software engineers have learned from experience is that it is important to separate the definition of what a system should do (a specification) from the definition of how it should work (a design). This separation allows the two specialist groups (testers for specifications and designers for design) to work independently so that we can later check that they have arrived at the same place, i.e. they have together built a system and tested that it works according to its specification.

If we set up test cases so that we check that desired behavior actually occurs then we are acting independently of the developers. If they have misunderstood the specification or chosen to change it in some way without telling anyone then their implementation will deliver behavior that is different from what the model or specification said the system behavior should be. Our test, based solely on the specification, will therefore fail and we will have uncovered a problem.

It may be noted that not all systems are defined by a detailed formal specification. In some cases the model we use may be quite informal. If there is no specification at all, the tester may have to build a model of the proposed system, perhaps by interviewing key stakeholders to understand what their expectations are. However formal or informal the model is, and however it is built, it provides a test basis from which we can generate tests systematically.

Remember that the specification can contain non-functional elements as well as functions; topics such as reliability, usability and performance are examples. These need to be systematically tested as well.

What we need, then, are techniques that can explore the specified behavior systematically and thoroughly in a way that is as efficient as we can make it. In addition, we use what we know about software to ‘home in’ on problems; each of the test case design techniques is based on some simple principles that arise from what we know in general about software behavior.

<<<<<< =================== >>>>>>

Q. 40: What are the different types of specification-based techniques?

We have following five major specification-based techniques:

1) Equivalence partitioning

2) Boundary value analysis

3) Decision table testing

4) State transition testing

5) Use case testing

Q. 41: What is the purpose of Equivalence partitioning the specification-based test case design technique

Equivalence partitioning is based on a very simple idea: it is that in many cases the inputs to a program can be ‘chunked’ into groups of similar inputs.

For example, a program that accepts integer values can accept as valid any input that is an integer (i.e. a whole number) and should reject anything else (such as a real number or a character). The range of integers is infinite, though the computer will limit this to some finite value in both the negative and positive directions (simply because it can only handle numbers of a certain size; it is a finite machine). Let us take an example, that the program accepts any value between −10,000 and +10,000 (computers actually represent numbers in binary form, which makes the numbers look much less like the ones we are familiar with, but we will stick to a familiar representation).

If we imagine a program that separates numbers into two groups according to whether they are positive or negative the total range of integers could be split into three ‘partitions’: the values that are less than zero; zero; and the values that are greater than zero. Each of these is known as an ‘equivalence partition’ because every value inside the partition is exactly equivalent to any other value as far as our program is concerned. So if the computer accepts −2,905 as a valid negative integer we would expect it also to accept −3. Similarly, if it accepts 100 it should also accept 2,345 as a positive integer.

Note that we are treating zero as a special case. We could, if we chose to, include zero with the positive integers, but my rudimentary specification did not specify that clearly, so it is really left as an undefined value (and it is not untypical to find such ambiguities or undefined areas in specifications). It often suits us to treat zero as a special case for testing where ranges of numbers are involved; we treat it as an equivalence partition with only one member. So we have three valid equivalence partitions in this case.

The equivalence partitioning technique takes advantage of the properties of equivalence partitions to reduce the number of test cases we need to write. Since all the values in an equivalence partition are handled in exactly the same way by a given program, we need only test one of them as a representative of the partition. In the example given, then, we need any positive integer, any negative integer and zero. We generally select values somewhere near the middle of each partition, so we might choose, say, −5,000, 0 and 5,000 as our representatives. These three test inputs would exercise all three partitions and the theory tells us that if the program treats these three values correctly it is very likely to treat all of the other values, all 19,998 of them in this case, correctly.

The partitions we have identified now are called valid equivalence partitions because they partition the collection of valid inputs, but there are other possible inputs to this program that would not be valid - real numbers, for example. We also have two input partitions of integers that are not valid: integers less than −10,000 and integers greater than 10,000. We should test that the program does not accept these, which is just as important as the program accepting valid inputs.

Non-valid partitions are also important to test. If you think about the example we have been using you will soon recognize that there are far more possible non-valid inputs than valid ones, since all the real numbers (e.g. numbers containing decimals) and all characters are non-valid in this case. It is generally the case that there are far more ways to provide incorrect input than there are to provide correct input; as a result, we need to ensure that we have tested the program against the possible non-valid inputs. Here again equivalence partitioning comes to our aid: all real numbers are equally non-valid, as are all alphabetic characters. These represent two non-valid partitions that we should test, using values such as 9.45 and ‘r’ respectively. There will be many other possible non-valid input partitions, so we may have to limit the test cases to the ones that are most likely to crop up in a real situation.

<<<<<< =================== >>>>>>

Q. 42: Describe some example of Equivalence Partitions

Valid input: integers in the range 100 to 999.

# Valid partition: 100 to 999 inclusive.

# Non-valid partitions: less than 100, more than 999, real (decimal) numbers and non-numeric characters.

Valid input: names with up to 20 alphabetic characters.

# Valid partition: strings of up to 20 alphabetic characters.

# Non-valid partitions: strings of more than 20 alphabetic characters, strings containing non-alphabetic characters.

<<<<<< =================== >>>>>>

Q. 43: What is the purpose of Boundary Value Analysis the specification-based test case design technique

There is a common mistake that programmers make is that errors tend to cluster around boundaries. For example, if a program should accept a sequence of numbers between 1 and 10, the most likely fault will be that values just outside this range are incorrectly accepted or that values just inside the range are incorrectly rejected. In the programming world these faults coincide with particular programming structures such as the number of times a program loop is executed or the exact point at which a loop should stop executing.

This works well with our equivalence partitioning idea because partitions must have boundaries. A partition of integers between 1 and 99, for instance, has a lowest value, 1, and a highest value, 99. These are called boundary values. Actually they are called valid boundary values because they are the boundaries on the inside of a valid partition. What about the values on the outside? Yes, they have boundaries too. So the boundary of the non-valid values at the lower end will be zero because it is the first value you come to when you step outside the partition at the bottom end. (You can also think of this as the highest value inside the non-valid partition of integers that are less than one, of course.) At the top end of the range we also have a non-valid boundary value, 100.

This is the boundary value technique, more or less. For most practical purposes the boundary value analysis technique needs to identify just two values at each boundary. For reasons that need not detain us here there is an alternative version of the technique that uses three values at each boundary.

For this variant, which is the one documented in BS 7925-2, we include one more value at each boundary when we use boundary value analysis: the rule is that we use the boundary value itself and one value (as close as you can get) either side of the boundary.

So, in this case lower boundary values will be 0, 1, 2 and upper boundary values will be 98, 99, 100.

What does ‘as close as we can get’ mean? It means take the next value in sequence using the precision that has been applied to the partition. If the numbers are to a precision of 0.01, for example, the lower boundary values would be 0.99, 1.00, 1.01 and the upper boundary values would be 98.99, 99.00, 99.01.

<<<<<< =================== >>>>>>

Q. 44: Describe some example of Boundary Values

1) The boiling point of water: The boundary is at 100 degrees Celsius, so for the 3 Value Boundary approach the boundary values will be 99 degrees, 100 degrees, 101 degrees—unless you have a very accurate digital thermometer, in which case they could be 99.9 degrees, 100.0 degrees, 100.1 degrees. For the 2 value approach the corresponding values would be 100 and 101.

2) Exam pass: If an exam has a pass boundary at 40 per cent, merit at 60 per cent and distinction at 80 per cent the 3 value boundaries would be 39, 40, 41 for pass, 59, 60, 61 for merit, 79, 80, 81 for distinction. It is unlikely that marks would be recorded at any greater precision than whole numbers. The 2 value equivalents would be 39 and 40, 59 and 60, and 79 and 80 respectively.

<<<<<< =================== >>>>>>

Q. 45: What is the purpose of Decision Table Testing the specification-based test case design technique?

Specifications generally contain business rules to define the functions of the system and the conditions under which each function operates. Individual decisions are usually simple, but the overall effect of these logical conditions can become quite complex.

As testers we need to be able to assure ourselves that every combination of these conditions that might occur has been tested, so we need to capture all the decisions in a way that enables us to explore their combinations. The mechanism usually used to capture the logical decisions is called a decision table.

A decision table lists all the input conditions that can occur and all the actions that can arise from them. These are structured into a table as rows, with the conditions at the top of the table and the possible actions at the bottom.

Business rules, which involve combinations of conditions to produce some combination of actions, are arranged across the top. Each column therefore represents a single business rule (or just ‘rule’) and shows how input conditions combine to produce actions. Thus each column represents a possible test case, since it identifies both inputs and expected outputs. The schematic structure of a decision table is shown in the following table.

	Business rule 1	Business rule 2	Business rule 3
Condition 1	T	F	T
Condition 2	T	T	T

Condition 3	T	"-"	F
Action 1	Y	N	Y
Action 2	N	Y	Y

Business rule 1 requires all conditions to be true to generate action 1. Business rule 2 results in action 2 if condition 1 is false and condition 2 is true but does not depend on condition 3. Business rule 3 requires conditions 1 and 2 to be true and condition 3 to be false.

In reality the number of conditions and actions can be quite large, but usually the number of combinations producing specific actions is relatively small. For this reason we do not enter every possible combination of conditions into our decision table, but restrict it to those combinations that correspond to business rules - this is called a limited entry decision table to distinguish it from a decision table with all combinations of inputs identified.

Q. 46: Describe some example of Decision Tables

A supermarket has a loyalty scheme that is offered to all customers. Loyalty cardholders enjoy the benefits of either additional discounts on all purchases (rule 3) or the acquisition of loyalty points (rule 4), which can be converted into vouchers for the supermarket or to equivalent points in schemes run by partners. Customers without a loyalty card receive an additional discount only if they spend more than $100 on any one visit to the store (rule 2), otherwise only the special offers offered to all customers apply (rule 1).

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**
Condition:				
Customer without loyalty card	T	T	F	F
Customer with loyalty card	F	F	T	T
Extra discount selected	"-"	"-"	T	F
Spend > $100	F	T	"-"	"-"
Actions:				

No discount	T	F	F	F
Extra discount	F	T	T	F
Loyalty points	F	F	F	T

From the decision table we can determine test cases by setting values for the conditions and determining the expected output, e.g. from rule 1 we could input a normal customer with a $50 transaction and check that no discount was applied. The same customer with a $150 transaction (rule 2) should attract a discount. Thus we can see that each column of the decision table represents a possible test case.

<<<<<< =================== >>>>>>

Q. 47: What is the purpose of State Transition Testing the specification-based test case design technique

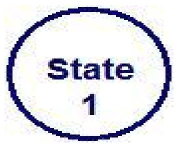
The decision table testing, is useful in systems where combinations of input conditions produce various actions.

State Transition Testing is also a similar technique, but this is concerned with systems in which outputs are triggered by changes to the input conditions, or changes of 'state'; in other words, behavior depends on current state and past state, and it is the transitions that trigger system behavior.

This technique is known as state transition testing or that the main diagram used in the technique is called a state transition diagram.

A state transition diagram is a representation of the behavior of a system. It is made up from just two symbols.

The first symbol is a circle as shown below



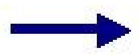
This is the symbol for a state.

A state is just what it says it is:

1) The system is 'static'
2) The system is in a stable condition from which it will only change if it is stimulated by an event of some kind.

For example, a TV stays 'on' unless you turn it 'off'; a multifunction watch tells the time unless you change the mode.

The second symbol is an arrow



This is the symbol for a transition, i.e. a change from one state to another. The state change will be triggered by an event (e.g. pressing a button or switching a switch). The transition will be labelled with the event that caused it and any action that arises from it. So we might have ‘mode button pressed’ as an event and ‘presentation changes’ as the action.

Usually the Start State will have a double arrowhead pointing to it. Often the Start State is obvious anyway.

If we have a state transition diagram representation of a system we can analyze the behavior in terms of what happens when a transition occurs.

Transitions are caused by events and they may generate outputs and/or changes of state. An event is anything that acts as a trigger for a change; it could be an input to the system, or it could be something inside the system that changes for some reason, such as a database field being updated.

In some cases an event generates an output, in others the event changes the system's internal state without generating an output, and in still others an event may cause an output and a change of state. What happens for each change is always deducible from the state transition diagram.

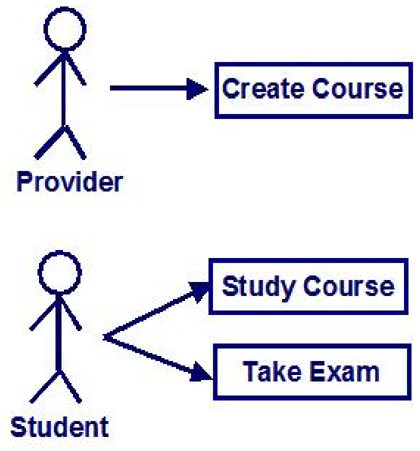
<<<<<< =================== >>>>>>

Q. 48: What is the purpose of Use Case testing the specification-based test case design technique

Use cases are one way of specifying functionality as business scenarios or process flows. They capture the individual interactions between ‘actors’ and the system. An actor represents a particular type of user and the use cases capture the interactions that each user takes part in to produce some output that is of value. Test cases based on use cases at the business process level, often called scenarios, are particularly useful in exercising business rules or process flows and will often identify gaps or weaknesses in these that would not be found by exercising individual components in isolation. This makes use case testing very effective in defining acceptance tests because the use cases represent actual likely use.

Use cases may also be defined at the system level, with preconditions that define the state the system needs to be in on entry to a use case to enable the use case to complete successfully, and post-conditions that define the state of the system on completion of the use case. Use cases typically have a mainstream path, defining the expected behavior, and one or more alternative paths related to such aspects as error conditions. Well-defined use cases can therefore be an excellent basis for system level testing, and they can also help to uncover integration defects caused by incorrect interaction or communication between components.

In practice, writing a test case to represent each use case is often a good starting point for testing, and use case testing can be combined with other specification-based testing.



The above figure alone does not provide enough detail for testing, so we need some textual description of the processes involved as well.

Use case testing has the major benefit that it relates to real user processes, so it offers an opportunity to exercise a complete process flow. The principles applied elsewhere can be applied here: first test the highest priority (highest value) use cases by taking typical examples; then exercise some attempts at incorrect process flows; and then exercise the boundaries.

<<<<<< =================== >>>>>>

Q. 49: What is the purpose of Structure-Based (White-Box) Testing Techniques?

Structure-based test techniques are used to explore system or component structures at several levels. At the component level, the structures of interest will be program structures such as decisions; at the integration level we may be interested in exploring the way components interact with other components (in what is usually termed a calling structure); at the system level we may be interested in how users will interact with a menu structure.

All these are examples of structures and all may be tested using white-box test case design techniques. Instead of exercising a component or system to see if it functions correctly white-box tests focus on ensuring that particular elements of the structure itself are correctly exercised. For example, we can use structural testing techniques to ensure that each statement in the code of a component is executed at least once. At the component level, where structure-based testing is most commonly used, the test case design techniques involve generating test cases from code, so we need to be able to read and analyze the code.

As code analysis and structure-based testing at the component level are mostly done by specialist tools, but a knowledge of the techniques is still valuable. You may wish to run simple test cases on code to ensure that it is basically sound before you begin detailed functional testing, or you may want to interpret test results from programmers to ensure that their testing adequately exercises the code.

<<<<<< =================== >>>>>>

Q. 50: What is a Pseudo Code & what are its advantages?

In ISTQB Foundation-level examination the term ‘code’ will always mean pseudo code. Pseudo code is a much more limited language than any real programming language but it enables designers to create all the main control structures needed by programs. It is sometimes used to document designs before they are coded into a programming language.

Real programming languages have a wide variety of forms and structures so many that we could not adequately cover them all. The advantage of pseudo code in this respect is that it has a simple structure.

Q. 51: What is the general program structure of a pseudo code?

Code can be of two types, executable and non-executable.

Executable code instructs the computer to take some action; non-executable code is used to prepare the computer to do its calculations but it does not involve any actions.

For example, reserving space to store a calculation (this is called a declaration statement) involves no actions. In pseudo code non-executable statements will be at the beginning of the program; the start of the executable part is usually identified by BEGIN, and the end of the program by END.

So we get the following structure:

```
1 Non-executable statements
2 BEGIN
3
4 Executable statements
5
6 END
```

If we were counting executable statements we would count lines 2, 4 and 6. Line 1 is not counted because it is non-executable. Lines 3 and 5 are ignored because they are blank.

If there are no non-executable statements there may be no BEGIN or END either, but there will always be something separating non-executable from executable statements where both are present.

<<<<<< =================== >>>>>>

Q. 52: In how many ways we can structure the executable code?

Once we have a picture of an overall program structure we can look inside the code. There are only following three ways that executable code can be structured.

1) Sequence: The first is simple & here the statements are exercised one after the other as they appear on the page.

Following is an example of a purely sequential program

```
1 Read A
2 Read B
3 C = A + B
```

The BEGIN and END have been omitted in this case since there were no non-executable statements; this is not strictly correct but is common practice, so it is wise to be aware of it and remember to check whether there are any non-executable statements when you do see BEGIN and END in a program. The computer would execute those three statements in sequence, so it would read (input) a value into A (this is just a name for a storage location), then read another value into B, and finally add them together and put the answer into C.

2) Selection: In this case the computer has to decide if a condition (known as a Boolean condition) is true or false. If it is true the computer takes one route, and if it is false the computer takes a different route. Selection structures therefore involve decisions.

Following is an example of a Selection program

```
1 IF P > 3
2 THEN
3 X = X + Y
4 ELSE
5 X = X – Y
6 ENDIF
```

Here we ask the computer to evaluate the condition P > 3, which means compare the value that is in location P with 3. If the value in P is greater than 3 then the condition is true; if not, the condition is false. The computer then selects which statement to execute next. If the condition is true it will execute the part labeled THEN, so it executes line 3. Similarly if the condition is false it will execute line 5. After it has executed either line 3 or line 5 it will go to line 6, which is the end of the selection (IF THEN ELSE) structure. From there it will continue with the next line in sequence.

There may not always be an ELSE part, as below:

```
1 IF P > 3
2 THEN
3 X = X + Y
4 ENDIF
```

In this case the computer executes line 3 if the condition is true, or moves on to line 4 (the next line in sequence) if the condition is false.

3) Iteration: It simply involves the computer exercising a chunk of code more than once; the number of times it exercises the chunk of code depends on the value of a condition (just as in the selection case).

Iteration structures are called loops. The most common loop is known as a DO WHILE (or WHILE DO) loop and is illustrated below:

```
1 X = 15
2 Count = 0
3 WHILE X < 20 DO
4 X = X + 1
5 Count = Count + 1
6 END DO
```

As with the selection structures there is a decision. In this case the condition that is tested at the decision is X < 20. If the condition is true the program ‘enters the loop’ by executing the code between DO and END DO. In this case the value of X is increased by one and the value of Count is increased by one. When this is done the program goes back to line 3 and repeats the test. If X < 20 is still true the program ‘enters the loop’ again. This continues as long as the condition is true.

If the condition is false the program goes directly to line 6 and then continues to the next sequential instruction. In the program fragment above the loop will be executed five times before the value of X reaches 20 and causes the loop to terminate. The value of Count will then be 5.

There is another variation of the loop structure known as a REPEAT UNTIL loop. It looks like this:

```
1 X = 15
2 Count = 0
3 REPEAT
4 X = X + 1
5 Count = Count + 1
6 UNTIL X = 20
```

The difference from a DO WHILE loop is that the condition is at the end, so the loop will always be executed at least once. Every time the code inside the loop is executed the program checks the condition. When the condition is true the program continues with the next sequential instruction. The outcome of this REPEAT UNTIL loop will be exactly the same as the DO WHILE loop above.

<<<<<< =================== >>>>>>

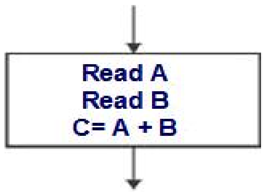
Q. 53: What is the purpose of creating flow charts?

After we are able to write code we go a step further and create a visual representation of the structure that is much easier to work with. The simplest visual structure to draw is the flow chart, which has only two symbols.

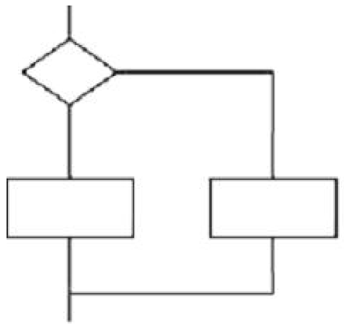
Rectangles represent sequential statements and diamonds represent decisions. More than one sequential statement can be placed inside a single rectangle as long as there are no decisions in the sequence. Any decision is represented by a diamond, including those associated with loops.

To create a flow chart representation of a complete program all we need to do is to connect together all the different bits of structure.

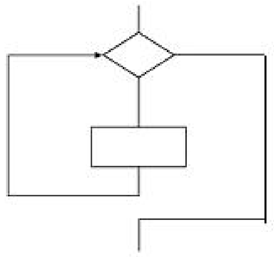
Example of a Flow chart for a sequential program



Example of a Flow chart for a selection (decision) structure



Example of a Flow chart for an iteration (loop) structure



<<<<<< =================== >>>>>>

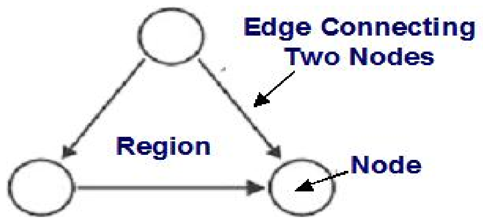
Q. 54: What is the purpose of making Control Flow Graphs

A control flow graph provides a method of representing the decision points and the flow of control within a piece of code, so it is just like a flow chart except that it only shows decisions. A control flow graph is produced by looking only at the statements affecting the flow of control.

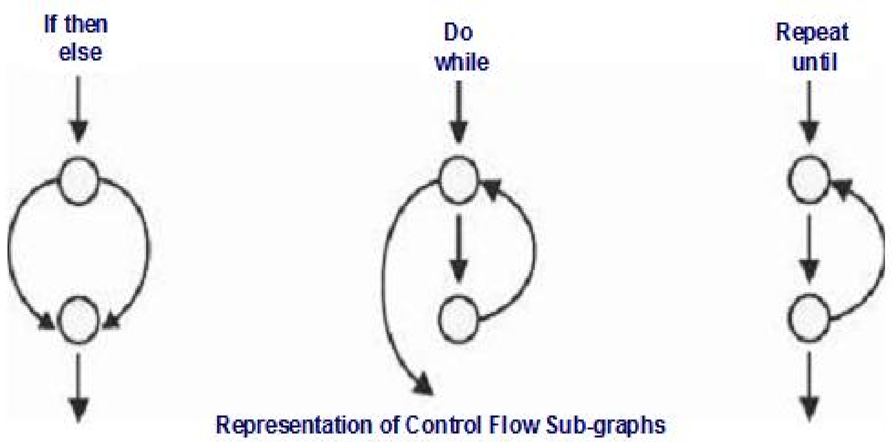
The graph itself is made up of two symbols: 1) Nodes and 2) Edges.

1) A node represents any point where the flow of control can be modified (i.e. decision points), or the points where a control structure returns to the main flow (e.g. END WHILE or ENDIF).

2) An edge is a line connecting any two nodes. The closed area contained within a collection of nodes and edges, as shown in the diagram, is known as a region.



We can draw 'sub-graphs' to represent individual structures. For a flow graph the representation of sequence is just a straight line, since there is no decision to cause any branching.



Representation of Control Flow Sub-graphs

The sub-graphs show what the control flow graph would look like for the program structures we are already familiar with.

<<<<<< =================== >>>>>>

Q. 55: What are the basic steps involved in drawing a control flow graph?

Step-1: Analyze the component to identify all control structures, i.e. all statements that can modify the flow of control, ignoring all sequential statements.

Step-2: Add a node for any decision statement.

Step-3: Expand the node by substituting the appropriate sub-graph representing the structure at the decision point. Any chunk of code can be represented by using these sub-graphs.

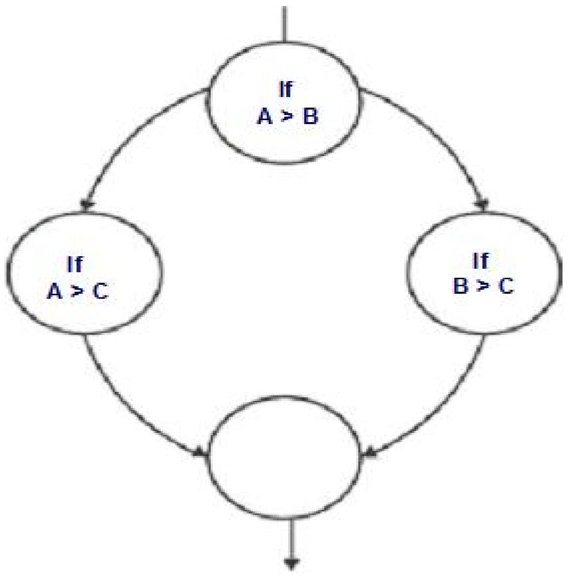
Refer the following example code.

Step 1: breaks the code into statements and identifies the control structures, ignoring the sequential statements, in order to identify the decision points; these are highlighted below.

```
1 Program MaxandMean
2
3 A, B, C, Maximum: Integer
4 Mean: Real
5
6 Begin
7
8 Read A
9 Read B
10 Read C
11 Mean = (A + B + C)/3
12
13 If A > B
14 Then
15 If A > C
16 Then
17 Maximum = A
18 Else
19 Maximum = C
```

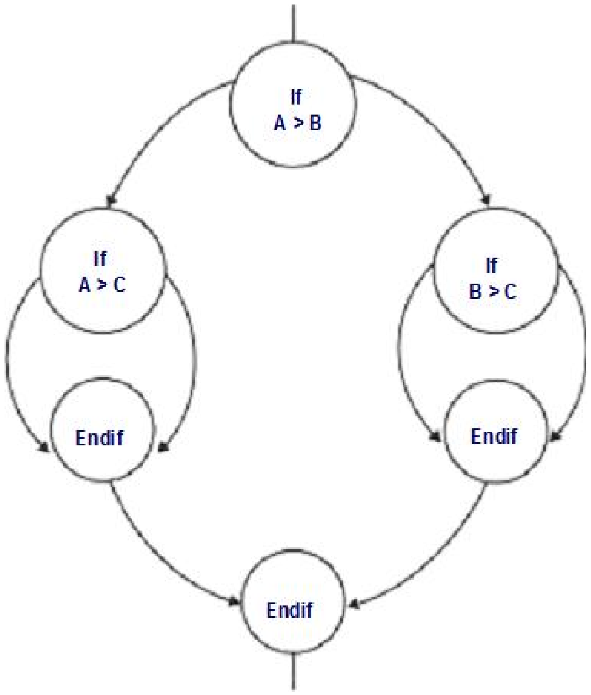
```
Endif
Else
If B > C
Then
Maximum = B
Else
Maximum = C
Endif
Endif

Print ("Mean of A, B and C is ", Mean)
Print ("Maximum of A, B, C is ", Maximum)

End
```

Step 2: Adds a node for each branching or decision statement as shown in following figure.



Step 3: Expands the nodes by substituting the appropriate sub-graphs as shown in following

Fugure



Q. 56: What is the purpose of Statement Testing and Coverage?

Statement testing is testing aimed at exercising programming statements. If we aim to test every executable statement we call this full or 100 per cent statement coverage. If we exercise half the executable statements this is 50 per cent statement coverage, and so on. Remember: we are only interested in executable statements, so we do not count non-executable statements at all when we are measuring statement coverage.

Why should we measure the statement coverage?

It is a very basic measure that testing has been (relatively) thorough. After all, a suite of tests that had not exercised all of the code would not be considered complete. Actually, achieving 100 per cent statement coverage does not tell us very much, and there are much more rigorous coverage measures that we can apply, but it provides a baseline from which we can move on to more useful coverage measures.

<<<<<< =================== >>>>>>

Q. 57: What is the purpose of Decision Testing and Coverage?

Decision testing aims to ensure that the decisions in a program are adequately exercised. Decisions, as you know, are part of selection and iteration structures; we see them in IF THEN ELSE constructs and in DO WHILE or REPEAT UNTIL loops. To test a decision we need to exercise it when the associated condition is true and when the condition is false; this guarantees that both exits from the decision are exercised.

As with statement testing, decision testing has an associated coverage measure and we normally aim to achieve 100 per cent decision coverage. Decision coverage is measured by counting the number of decision outcomes exercised (each exit from a decision is known as a decision outcome) divided by the total number of decision outcomes in a given program. It is usually expressed as a percentage.

The usual starting point is a control flow graph, from which we can visualize all the possible decisions and their exit paths.

Let us consider the following example.

```
Program Check

Count, Sum, Index: Integer

Begin

Index = 0
Sum = 0
Read (Count)
Read (New)

While Index <= Count
Do
If New < 0
Then
Sum = Sum + 1
Endif
Index = Index + 1
Read (New)
Enddo

Print ("There were", Sum, "negative numbers in the input stream")

End
```

This program has a WHILE loop in it. There is a golden rule about WHILE loops. If the condition at the WHILE statement is true when the program reaches it for the first time then any test case will exercise that decision in both directions because it will eventually be false when the loop terminates. For example, as long as Index is less than Count when the program reaches the loop for the first time, the condition will be true and the loop will be entered. Each time the program runs through the loop it will increase the value of Index by one, so eventually Index will reach the value of Count and pass it, at which stage the condition is false and the loop will not be entered. So the decision at the start of the loop is exercised through both its true exit and its false exit by a single test case. This makes the assumption that the logic of the loop is sound, but we are assuming that we are receiving this program from the developers who will have debugged it.

Now all we have to do is to make sure that we exercise the If statement inside the loop through both its true and false exits. We can do this by ensuring that the input stream has both negative and positive numbers in it.

For example, a test case that sets the variable Count to 5 and then inputs the values 1, 5, −2, −3, 6 will exercise all the decisions fully and provide us with 100 per cent decision coverage. Note that this is considered to be a single test case, even though there is more than one value for the variable New, because the values are all input in a single execution of the program. This example does not provide the smallest set of inputs that would achieve 100 per cent decision coverage, but it does provide a valid example.

Although loops are a little more complicated to understand than programs without loops, they can be easier to test once you get the hang of them.

<<<<<< =================== >>>>>>

Q. 58: What do we mean by experience-based techniques?

Experience-based techniques are those that you need to deploy when there is no adequate specification from which to derive specification-based test cases or no time to run the full structured set of tests.

They use the users' and the testers' experience to determine the most important areas of a system and to exercise these areas in ways that are both consistent with expected use (and abuse) and likely to be the sites of errors - this is where the experience comes in. Even when specifications are available it is worth supplementing the structured tests with some that you know by experience have found defects in other similar systems.

Techniques range from the simplistic approach of ad hoc testing or error guessing through to the more sophisticated techniques such as exploratory testing, but all tap the knowledge and experience of the tester rather than systematically exploring a system against a written specification.

<<<<<< =================== >>>>>>

Q. 59: What are the different types of experience-based testing techniques?

There are mainly two techniques under this category;

1) Error Guessing: It is a very simple technique that takes advantage of a tester's skill, intuition and experience with similar applications to identify special tests that may not be easy to capture by the more formal techniques. When applied after systematic techniques, error guessing can add another value in identifying and exercising test cases that target known or suspected weaknesses or that simply address aspects of the application that have been found to be problematical in the past.

The main drawback of error guessing is its varying effectiveness, depending as it does on the experience of the tester deploying it. However, if several testers and/or users contribute to constructing a list of possible errors and tests are designed to attack each error listed, this weakness can be effectively overcome. Another way to make error guessing more structured is by the creation of defect and failure lists. These lists can use available defect and failure data (where this exists) as a starting point, but the list can be expanded by using the testers' and users' experience of why the application under test in particular is likely to fail. The defect and failure list can be used as the basis of a set of tests that are applied after the systematic techniques have been used. This systematic approach is known as fault attack.

2) Exploratory Testing: It is a technique that combines the experience of testers with a structured approach to testing where specifications are either missing or inadequate and where there is severe time pressure. It exploits concurrent test design, test execution, test logging and learning within time-boxes and is structured around a test charter containing test objectives. In this way exploratory testing maximizes the amount of testing that can be achieved within a limited time frame, using test objectives to maintain focus on the most important areas.

<<<<<< =================== >>>>>>

Q. 60: How do we decide which is the best experience-based testing techniques?

Following are some simple thumb rules:

1) Always make functional testing the first priority. It may be necessary to test early code products using structural techniques, but we only really learn about the quality of software when we can see what it does.

2) When basic functional testing is complete that is a good time to think about test coverage. Have you exercised all the functions, all the requirements, all the code? Coverage measures defined at the beginning as exit criteria can now come into play. Where coverage is inadequate extra tests will be needed.

3) Use structural methods to supplement functional methods where possible. Even if functional coverage is adequate, it will usually be worth checking statement and decision coverage to ensure that enough of the code has been exercised during testing.

4) Once systematic testing is complete there is an opportunity to use experience-based techniques to ensure that all the most important and most error-prone areas of the software have been exercised. In some circumstances, such as poor specifications or time pressure, experience -based testing may be the only viable option.

ISTQB Foundation Level Exam Crash Course Part-13

This is Part 13 of 30 containing 5 Questions (Q. 61 to 65) with detailed explanation as expected in ISTQB Foundation Level Exam Latest Syllabus updated in 2010

Deep study of these 150 questions shall be of great help in getting success in ISTQB Foundation Level Exam

Q. 61: How do we take a decision of which experience based test technique to use?

It is not a simple task to decide in favor of any particular technique. However following factors are helpful

1) Type of system

2) Regulatory standards

3) Customer or contractual requirements

4) Level of risk

5) Type of risk

6) Test objectives

7) Documentation available

8) Knowledge of the testers

9) Time and budget

10) Development life cycle

11) Use case models

12) Experience of type of defects found

One or more of the above factors may be important on any given occasion. Some factors leave no room for any selection like the following:

# Regulatory or contractual requirements leave the tester with no choice.

# Test objectives, where they relate to exit criteria such as test coverage, may also lead to mandatory techniques.

# Where documentation is not available, or where time and budget are limited, the use of experience-based techniques may be favored.

# All others provide pointers within a broad framework: level and type of risk will push the tester towards a particular approach, where high risk is a good reason for using systematic techniques;

# Knowledge of testers, especially where this is limited, may narrow down the available choices;

# The type of system and the development life cycle will encourage testers to lean in one direction or another depending on their own particular experience.

There are few clear-cut cases, and the exercise of sound judgement in selecting appropriate techniques is a mark of a good test manager or team leader.

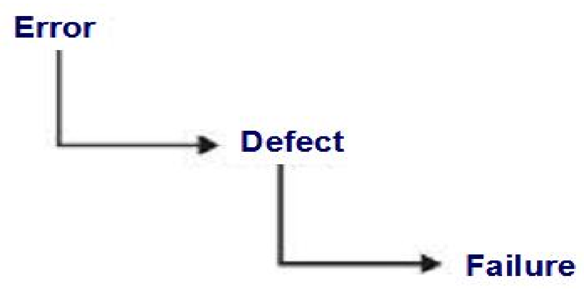
<<<<<< =================== >>>>>>

Q. 62: What are the reasons of failure of software systems?

The people who design the systems make mistakes. People make mistakes because they are fallible, but there are also many pressures that make mistakes more likely. Pressures such as deadlines, complexity of systems and organizations, and changing technology all bear down on designers of systems and increase the likelihood of errors in specifications, in designs and in software code. These errors are where major system failures usually begin.

If a document with an error in it is used to specify a component the component will be faulty and will probably exhibit incorrect behavior. If this faulty component is built into a system the system may fail. While failure is not always guaranteed, it is likely that errors in specifications will lead to faulty components and faulty components will cause system failure.

An error (or mistake) leads to a defect, which can cause an observed failure as shown in the following figure.



There are other reasons why systems fail. Environmental conditions such as the presence of radiation, magnetism, electronic fields or pollution can affect the operation of hardware and firmware and lead to system failure.

If we want to avoid failure we must either avoid errors and faults or find them and rectify them. Testing can contribute to both avoidance and rectification, as we will see when we have looked at the testing process in a little more detail.

If we wish to influence errors with testing we need to do the following

a) Begin testing as soon as we begin making errors

b) Begin testing right at the beginning of the development process

c) Continue testing until we are confident that there will be no serious system failures

d) Continue testing right up-to the end of the development process.

<<<<<< =================== >>>>>>

Q. 63: What are the different effects or harms of an incorrect software?

1) To people: e.g. by causing an aircraft crash in which people die, or by causing a hospital life support system to fail

2) To companies: e.g. by causing incorrect billing, which results in the company losing money

3) To the environment: e.g. by releasing chemicals or radiation into the atmosphere.

Software failures can sometimes cause all three of these at once. The scenario of a train carrying nuclear waste being involved in a crash has been explored to help build public confidence in the safety of transporting nuclear waste by train. A failure of the train's on-board systems or of the signaling system that controls the train's movements could lead to catastrophic results. This may not be likely but it is a possibility that could be linked with software failure. Software failures, then, can lead to:

a) Loss of money

b) Loss of time

c) Loss of business reputation

d) Injury

e) Death

<<<<<< =================== >>>>>>

Q. 64: What is the relation between Testing and Risk?

Risk is inherent in all software development. The system may not work or the project to build it may not be completed on time, for example. These uncertainties become more significant as the system complexity and the implications of failure increase. Intuitively, we would expect to test an automatic flight control system more than we would test a video game system. Why? Because the risk is greater.

There is a greater probability of failure in the more complex system and the impact of failure is also greater.

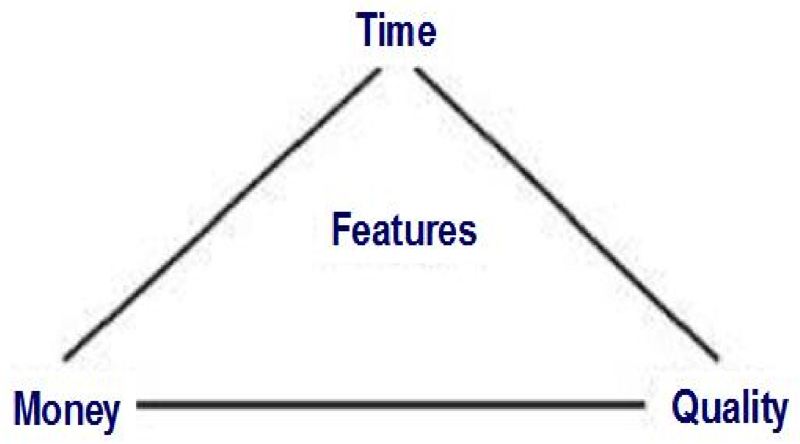
What we test, and how much we test it, must be related in some way to the risk. Greater risk implies more and better testing.

<<<<<< =================== >>>>>>

Q. 65: What is the relation between Testing and Quality?

Quality is hard to define. If a system meets its users' requirements that constitutes a good starting point.

The software development process, like any other, must balance competing demands for resources. If we need to deliver a system faster (i.e. in less time), for example, it will usually cost more. The items at the corners (or vertices) of the triangle of resources shown in the following figure are time, money and quality. These three affect one another, and also influence the features that are or are not included in the delivered software.



One role for testing is to ensure that key functional and non-functional requirements are examined before the system enters service and any defects are reported to the development team for rectification. Testing cannot directly remove defects, nor can it directly enhance quality. By reporting defects it makes their removal possible and so contributes to the enhanced quality of the system. In addition, the systematic coverage of a software product in testing allows at least some aspects of the quality of the software to be measured. Testing is one component in the overall quality assurance activity that seeks to ensure that systems enter service without defects that can lead to serious failures.

One role for testing is to ensure that key functional and non-functional requirements are examined before the system enters service and any defects are reported to the development team for rectification. Testing cannot directly remove defects, nor can it directly enhance quality. By reporting defects it makes their removal possible and so contributes to the enhanced quality of the system. In addition, the systematic coverage of a software product in testing allows at least some aspects of the quality of the software to be measured. Testing is one component in the overall quality assurance activity that seeks to ensure that systems enter service without defects that can lead to serious failures.

Q. 66: How much testing is enough, and how do we decide when to stop testing?

We know that we cannot test everything, even if we would wish to. We also know that every system is subject to risk of one kind or another and that there is a level of quality that is acceptable for a given system. These are the factors we will use to decide how much testing to do.

The most important aspect of achieving an acceptable result from a finite and limited amount of testing is prioritization. Do the most important tests first so that at any time you can be certain that the tests that have been done are more important than the ones still to be done. Even if the testing activity is cut in half it will still be true that the most important testing has been done. The most important tests will be those that test the most important aspects of the system: they will test the most important functions as defined by the users or sponsors of the system, and the most important non-functional behavior, and they will address the most significant risks.

The next most important aspect is setting criteria that will give you an objective test of whether it is safe to stop testing, so that time and all the other pressures do not confuse the outcome. These criteria, usually known as completion criteria, set the standards for the testing activity by defining areas such as how much of the software is to be tested.

Priorities and completion criteria provide a basis for planning. In the end, the desired level of quality and risk may have to be compromised, but our approach ensures that we can still determine how much testing is required to achieve the agreed levels and we can still be certain that any reduction in the time or effort available for testing will not affect the balance - the most important tests will still be those that have already been done whenever we stop.

<<<<<< =================== >>>>>>

Q. 67: What is the difference between Testing and Debugging?

Testing and debugging are different kinds of activity, both of which are very important. Debugging is the process that developers go through to identify the cause of bugs or defects in code and undertake corrections. Ideally some check of the correction is made, but this may not extend to checking that other areas of the system have not been inadvertently affected by the correction.

Testing, on the other hand, is a systematic exploration of a component or system with the main aim of finding and reporting defects. Testing does not include correction of defects - these are passed on to the developer to correct. Testing does, however, ensure that changes and corrections are checked for their effect on other parts of the component or system.

Effective debugging is essential before testing begins to raise the level of quality of the component or system to a level that is worth testing, i.e. a level that is sufficiently robust to enable rigorous testing to be performed. Debugging does not give confidence that the component or system meets its requirements completely.

Testing makes a rigorous examination of the behavior of a component or system and reports all defects found for the development team to correct. Testing then repeats enough tests to ensure that defect corrections have been effective. So both are needed to achieve a quality result.

<<<<<< =================== >>>>>>

Q. 68: What is the difference between Static Testing and Dynamic Testing?

Static testing is the term used for testing where the code is not exercised. Remember that failures often begin with a human error, namely a mistake in a document such as a specification. We need to test these because errors are much cheaper to fix than defects or failures. That is why testing should start as early as possible.

Static testing involves techniques such as reviews, which can be effective in preventing defects, e.g. by removing ambiguities and errors from specification documents; this is a topic in its own right.

Dynamic testing is the kind that exercises the program under test with some test data, so we speak of test execution in this context. The discipline of software testing encompasses both static and dynamic testing.

<<<<<< =================== >>>>>>

Q. 69: What is the impact of software errors detected at different stages of development?

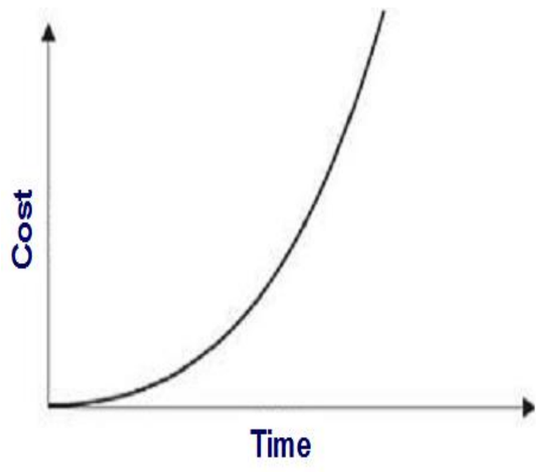
It is difficult to put figures on the relative costs of finding defects at different levels in the SDLC, following table is just an approximate indicator

Stage error is found	**Comparative cost**
Requirements	$1
Coding	$10
Program testing	$100
System testing	$1000

User acceptance testing	$100,00
Live running	$100,000

This is known as the cost escalation model.

What is undoubtedly true is that the graph of the relative cost of early and late identification/correction of defects rises very steeply as shown in the following graph.



The earlier a defect is found, the less it costs to fix.

The objectives of various stages of testing can be different. For example, in the review processes, we may focus on whether the documents are consistent and no errors have been introduced when the documents were produced. Other stages of testing can have other objectives. The important point is that testing has defined objectives.

<<<<<< =================== >>>>>>

Q. 70: What is Defect Clustering?

It is a fact that problems do occur in software. Once testing has identified the defects in a particular application, it is at first surprising that the spread of defects is not uniform. In a large application, it is often a small number of modules that exhibit the majority of the problems. This can be for a variety of reasons, some of which are:

1) System complexity.

2) Volatile code.

3) The effects of change upon change.

4) Development staff experience.

5) Development staff inexperience.

This is the application of the Pareto principle to software testing: approximately 80 per cent of the problems are found in about 20 per cent of the modules. It is useful if testing activity reflects this spread of defects, and targets areas of the application under test where a high proportion of defects can be found. However, it must be remembered that testing should not concentrate exclusively on these parts. There may be fewer defects in the remaining code, but testers still need to search diligently for them.

Q. 71: Is it helpful to execute the same set of tests continually?

Running the same set of tests continually will not continue to find new defects. Developers know that the test team always tests the boundaries of conditions, for example, so they will test these conditions before the software is delivered. This does not make defects elsewhere in the code less likely, so continuing to use the same test set will result in decreasing effectiveness of the tests. Using other techniques will find different defects.

For example, a small change to software could be specifically tested and an additional set of tests performed, aimed at showing that no additional problems have been introduced (this is known as regression testing). However, the software may fail in production because the regression tests are no longer relevant to the requirements of the system or the test objectives. Any regression test set needs to change to reflect business needs, and what are now seen as the most important risks.

<<<<<< =================== >>>>>>

Q. 72: What are the factors that determine the type of testing that is needed?

Different testing is necessary in different circumstances. A website where information can merely be viewed will be tested in a different way to an e-commerce site, where goods can be bought using credit/debit cards. We need to test an air traffic control system with more rigor than an application for calculating the length of a mortgage.

Risk can be a large factor in determining the type of testing that is needed. The higher the possibility of losses, the more we need to invest in testing the software before it is implemented.

For an e-commerce site, we should concentrate on security aspects. Is it possible to bypass the use of passwords? Can 'payment' be made with an invalid credit card, by entering excessive data into the card number? Security testing is an example of a specialist area, not appropriate for all applications. Such types of testing may require specialist staff and software tools.

<<<<<< =================== >>>>>>

Q. 73: What are the different elements of fundamental test process?

The fundamental test process consists of five sequential parts that encompass all aspects of testing as described in the following figure:

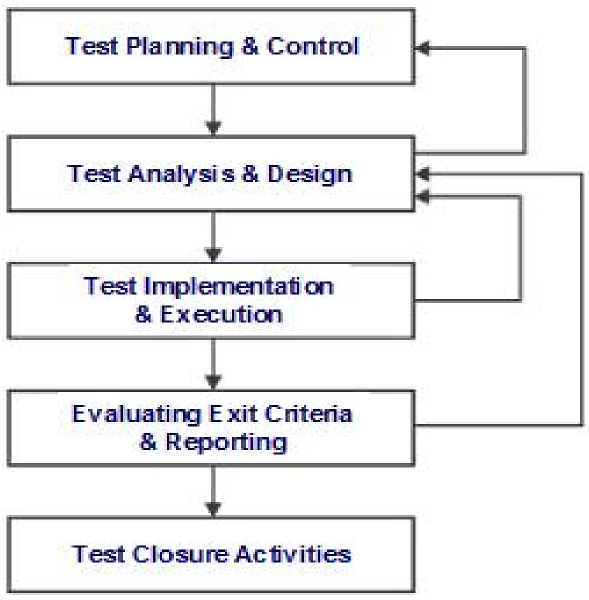
1) Planning and control.

2) Analysis and design.

3) Implementation and execution.

4) Evaluating exit criteria and reporting.

5) Test closure activities.



Although the main activities are in a broad sequence, they are not undertaken in a rigid way. An earlier activity may need to be revisited. A defect found in test execution can sometimes be resolved by adding functionality that was originally not present (either missing in error, or the new facilities are needed to make the other part correct). The new features themselves have to be tested, so even though implementation and execution are in progress, the 'earlier' activity of analysis and design has to be performed for the new features.

We sometimes need to do two or more of the main activities in parallel. Time pressure can mean that we begin test execution before all tests have been designed.

<<<<<< =================== >>>>>>

Q. 74: What is done during Test Planning and Control stage during the fundamental testing process?

Planning is determining what is going to be tested, and how this will be achieved. It is where we draw a map; how activities will be done; and who will do them. Test planning is also where we define the test completion criteria. Completion criteria are how we know when testing is finished.

Control, on the other hand, is what we do when the activities do not match up with the plans. It is the ongoing activity where we compare the progress against the plan. As progress takes place, we may need to adjust plans to meet the targets, if this is possible.

Therefore we need to undertake both planning and control throughout the testing activities. We plan at the outset, but as testing progresses, undertake monitoring and control activities (monitoring to measure what has happened, control to adjust future activities in the light of experience). Monitoring and control feed back into the continual activity of planning.

<<<<<< =================== >>>>>>

Q. 75: What is done during Test Analysis and Design stage during the fundamental testing process?

Analysis and design are concerned with the fine detail of what to test (test conditions), and how to combine test conditions into test cases, so that a small number of test cases can cover as many of the test conditions as possible.

The analysis and design stage is the bridge between planning and test execution. It is looking backward to the planning (schedules, people, what is going to be tested) and forward to the execution activity (test expected results, what environment will be needed).

A part of the design process needs to consider the test data that will be required for the test conditions and test cases that have been drawn up.

Test design involves predicting how the software under test should behave in a given set of circumstances. Sometimes the expected outcome of a test is trivial: when ordering books from an online book retailer, for instance, under no circumstances should money be refunded to the customer's card without intervention from a supervisor. If we do not detail expected outcomes before starting test execution, there is a real danger that we will miss the one item of detail that is vital, but wrong.

The main points of this activity are:

1) Reviewing requirements, architecture, design, interface specifications and other parts, which collectively comprise the test basis.

2) Analyzing test items, the specification, behavior and structure to identify test conditions and test data required.

3) Designing the tests, including assigning priority.

4) Determining whether the requirements and the system are testable.

5) Detailing what the test environment should look like, and whether there are any infrastructure and tools required.

6) Highlighting the test data required for the test conditions and test cases.

7) Creating bi-directional traceability between test basis and test cases.

Q. 76: What is done during Test Implementation and Execution stage during the fundamental testing process?

The test implementation and execution activity involves running tests, and this will include where necessary any set-up/tear-down activities for the testing. It will also involve checking the test environment before testing begins. Test execution is the most visible part of testing, but it is not possible without other parts of the fundamental test process. It is not just about running tests. The most important tests need to be run first. How do we know what are the most important tests to run? This is determined during the planning stages, and refined as part of test design.

One important aspect undertaken at this stage is combining test cases into an overall run procedure, so that test time can be utilized efficiently. Here the logical ordering of tests is important so that, where possible, the outcome of one test creates the preconditions for one or more tests that are later in the execution sequence.

As tests are run, their outcome needs to be logged, and a comparison made between expected results and actual results. Whenever there is a discrepancy between the expected and actual results, this needs to be investigated. If necessary a test incident should be raised. Each incident requires investigation, although corrective action will not be necessary in every case.

When anything changes (software, data, installation procedures, user documentation, etc.), we need to do two kinds of testing on the software. First of all, tests should be run to make sure that the problem has been fixed. We also need to make sure that the changes have not broken the software elsewhere. These two types are usually called re-testing and regression testing, respectively. I

n re-testing we are looking in fine detail at the changed area of functionality, whereas regression testing should cover all the main functions to ensure that no unintended changes have occurred. On a financial system, we should include end of day/end of month/end of year processing, for example, in a regression test pack.

Test implementation and execution is where the most visible test activities are undertaken, and usually have the following parts:

1) Developing and prioritizing test cases, creating test data, writing test procedures and, optionally, preparing test harnesses and writing automated test scripts.

2) Collecting test cases into test suites, where tests can be run one after another for efficiency.

3) Checking the test environment set-up is correct.

4) Running test cases in the determined order. This can be manually or using test execution tools.

5) Keeping a log of testing activities, including the outcome (pass/fail) and the versions of software, data, tools and scripts, etc..

6) Comparing actual results with expected results.

7) Reporting discrepancies as incidents with as much information as possible, including if possible causal analysis (code defect, incorrect test specification, test data error or test execution error).

8) Where necessary, repeating test activities when changes have been made following incidents raised. This includes re-execution of a test that previously failed in order to confirm a fix (re-testing), execution of a corrected test and execution of previously passed tests to check that defects have not been introduced (regression testing).

<<<<<< =================== >>>>>>

Q. 77: What is done during Evaluating Exit Criteria and Reporting stage during the fundamental testing process?

The exit criteria were defined during test planning and before test execution started.

At the end of test execution, the test manager checks to see if these have been met. If the criterion was that there would be 85 per cent statement coverage (i.e. 85 per cent of all executable statements have been executed and as a result of execution the figure is 75 per cent, there are two possible actions: change the exit criteria, or run more tests.

It is possible that even if the preset criteria were met, more tests would be required. Also, writing a test summary for stakeholders would say what was planned, what was achieved, highlight any differences and in particular things that were not tested.

The fourth stage of the fundamental test process, evaluating exit criteria, comprises of the following:

1) Checking whether the previously determined exit criteria have been met.

2) Determining if more tests are needed or if the specified exit criteria need amending.

3) Writing up the result of the testing activities for the business sponsors and other stakeholders.

<<<<<< =================== >>>>>>

Q. 78: What is done during Test Closure stage during the fundamental testing process?

Testing at this stage has finished. Test closure activities concentrate on making sure that everything is tidied away, reports written, defects closed, and those defects deferred for another phase clearly seen to be as such.

At the end of testing, the test closure stage is composed of the following:

1) Ensuring that the documentation is in order; what has been delivered is defined (it may be more or less than originally planned), closing incidents and raising changes for future deliveries, documenting that the system has been accepted.

2) Closing down and archiving the test environment, test infrastructure and testware used.

3) Passing over testware to the maintenance team.

4) Writing down the lessons learned from this testing project for the future, and incorporating lessons to improve the testing process.

<<<<<< =================== >>>>>>

Q. 79: What is the approach of test independence?

Testing can be more effective if it is not undertaken by the individuals who wrote the code, for the simple reason that the creator of anything (whether it is software or a work of art) has a special relationship with the created object. The nature of that relationship is such that flaws in the created object are rendered invisible to the creator.

For that reason it is important that someone other than the creator should test the object. This approach is called test independence. Following are the people who could test the software, listed in the order of increasing independence:

1) Those who wrote the code.

2) Members of the same development team.

3) Members of a different group (independent test team).

4) Members of a different company – may be a testing consultancy company or the testing project is outsourced.

Of course independence comes at a price; it is much more expensive to use a testing consultancy than to test a program oneself.

<<<<<< =================== >>>>>>

Q. 80: What are the code of ethics certified software testers are expected to follow?

Testers can have access to confidential or privileged information, and they are to treat any information with care and attention, and act responsibly to the owners of this information, employers and the wider public interest. Of course, anyone can test software, but the declaration of the code of conduct applies to those who have achieved software-testing certification.

The code of ethics applies to the following areas:

1) Public: Certified software testers shall consider the wider public interest in their actions.

2) Client and employer: Certified software testers shall act in the best interests of their client and employer (being consistent with the wider public interest).

3) Product: Certified software testers shall ensure that the deliverables they provide (for any products and systems they work on) meet the highest professional standards possible.

4) Judgement: Certified software testers shall maintain integrity and independence in their professional judgement.

5) Management: Certified software test managers and leaders shall subscribe to and promote and ethical approach to the management of software testing.

6) Profession: Certified software testers shall advance the integrity and reputation of the profession consistent with the public interest.

7) Colleagues: Certified software testers shall be fair to and supportive of their colleagues, and promote cooperation with software developers.

8) Self: Certified software testers shall participate in lifelong learning regarding the practice of their profession, and shall promote an ethical approach to the practice of the profession.

Q. 81: What are the key benefits of using testing tools?

The main benefit of using test tools is similar to the main benefit of automating any process. That is, the amount of time and effort spent performing routine, mundane, repetitive tasks is greatly reduced.

This time saved can be used to reduce the costs of testing or it can be used to allow testers to spend more time on the more intellectual tasks of test planning, analysis and design. In turn, this can enable more focused and appropriate testing to be done - rather than having many testers working long hours, running hundreds of tests.

Related to this is the fact that the automation of any process usually results in more predictable and consistent results. Similarly, the use of test tools provides more predictable and consistent results as human failings such as manual keying errors, misunderstandings, incorrect assumptions, forgetfulness, etc., are eliminated.

It also means that any reports or findings tend to be objective rather than subjective. For instance, humans often assume that something that seems reasonable is correct, when in fact it may not be what the system is supposed to do.

The widespread use of databases to hold the data input, processed or captured by the test tool, means that it is generally much easier and quicker to obtain and present accurate test management information, such as test progress, incidents found/fixed, etc.

<<<<<< =================== >>>>>>

Q. 82: What are the risks associated to the use of testing tools?

Most of the risks associated with the use of test tools are concerned with over-optimistic expectations of what the tool can do and a lack of appreciation of the effort required to implement and obtain the benefits that the tool can bring.

For example, consider the production environments of most organizations considering using test tools. They are unlikely to have been designed and built with test tools in mind. Therefore, assuming that you want a test environment to be a copy of production or at least as close to it as possible, you will also have a test environment that is not designed and built with test tools in mind.

Consider the test environments used by vendors to demonstrate their test tools. If you were the vendor would you design the environment to enable you to demonstrate the tool at its best or to demonstrate the shortcomings it may encounter in a typical test environment?

Therefore, unless detailed analysis and evaluation is done, it is likely that test tools will end up as something that seemed a good idea at the time but have been largely a waste of time and money.

After a test tool has been implemented and measurable benefits are being achieved, it is important to put in sufficient effort to maintain the tool, the processes surrounding it and the test environment in which it is used. Otherwise there is a risk that the benefits being obtained will decrease and the tool will become redundant. Additionally, opportunities for improving the way in which the tool is used could also be missed.

For example, the acquisition of various test tools from multiple vendors will require interfaces to be built or configured to import and export data between tools. Otherwise much time may be spent manually cutting and pasting data from one tool to another. If this is not done, then data inconsistencies and version control problems are likely to arise. Similar problems may arise when testing with third-party suppliers or as a result of mergers and acquisitions.

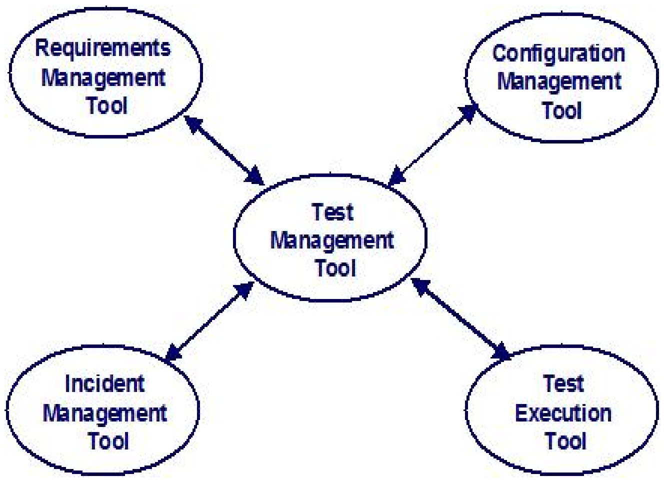
Maintenance effort will also be required to upgrade and re-configure tools so that they remain compliant with new platforms or operating systems.

<<<<<< =================== >>>>>>

Q. 83: What is the purpose of using Test Management Tools?

Test management tools provide support for various activities and tasks throughout the development life cycle. Some of these activities are supported by the provision of interfaces to other more specialist tools like test execution tools. Other test management tools provide fully integrated modules that provide some or all of the services / functions provided by more specialist tools like incident management or requirements management tools.

Following figure shows how a test management tool is the hub of a set of integrated test tools.



Test management tools provide an architecture for creating, storing and editing test procedures. These may be linked or traced to requirements, test conditions and risks. Such test procedures can then be prioritized or grouped together and scheduled so that they are run in the most effective and efficient order. Some test management tools allow dependencies to be recorded so that tests that will fail owing to a known defect can be highlighted and left unexecuted. This allows testers to be redirected to run the highest priority tests available rather than waste their time and the test data they have prepared on tests that are certain to fail.

Tests can be recorded as passed or failed and usually a test management tool provides an interface to an incident management tool so that an incident can be raised if the actual and expected results differ.

Test management tools can provide management information and reports on test procedures passed or failed. The amount of integration with other specialist tools is significant here. For instance, integration with requirements management tools allows reports to be produced on test progress against one or more requirements. Integration with incident management tools allows reports also to include analysis of defects against requirements.

Test management tools generally hold data in a database. This allows a large amount of reports and metrics to be produced. The metrics produced can be used as inputs to:

1) Test and project management to control the current project.

2) Estimates for future projects.

3) Identifying weaknesses or inefficiencies in the development or test process that can be subsequently investigated with the aim of improving them.

Test management information reports should be designed to meet the needs of project managers and other key users. It may be necessary to export data to a spreadsheet in order for it to be manipulated into the format required.

A test management tool can enable reuse of existing testware in future test projects.

<<<<<< =================== >>>>>>

Q. 84: What is the purpose of using Incident Management Tools?

Incident management tools are also known as defect management tools. These are one of the most widely used types of test tool. At a basic level incident management tools are used to perform two critical activities: creation of an incident report; and maintenance of details about the incident as it progresses through the incident life cycle.

The level of detail to be captured about the incident can be varied depending upon the characteristics of the tool itself and the way in which the incident management tool is configured and used by the test organization.

For example, the incident management tool could be configured so that lots of mandatory information is required in order to comply with industry or generic standards such as IEEE 1044. In addition, workflow rules may also be applied to ensure that the agreed incident life cycle is strictly applied, with incidents only able to be assigned to particular teams or users. Alternatively, the tool could be configured to require very limited mandatory information, with most fields being free format.

Incident management tools also use a database to store and manage details of incidents. This allows the incident to be categorized according to the values stored in appropriate fields. Such values will change during the incident life cycle as the incident is analyzed, debugged, fixed and re -tested. It is often possible to view the history of changes made to the incident.

The database structure also enables incidents to be searched and analyzed (using either filters or more complex SQL-type queries). This provides the basis for management information about incidents. Note that as the values held against each incident change, the management information will also change. Therefore users need to be aware of the danger of using outdated reports.

This data can also be used in conjunction with data held in test management tools when planning and estimating for future projects. It can also be analyzed to provide input to test process improvement projects.

Fields in the database structure normally include:

1) Priority (e.g. high, medium, low).

2) Severity (e.g. high, medium, low).

3) Assignee (the person to whom the incident is currently assigned, e.g. a developer for debugging, a tester to perform re-testing).

4) Status in the incident life cycle (e.g. New, Open, Fixed, Reopen, Closed).

Some test management tools include fully integrated incident management tools as part of their core product, whilst other incident management tools can be integrated with test management, requirements management and/or test execution tools. Such integration enables incidents to be input and traced back to test cases and requirements.

<<<<<< =================== >>>>>>

Q. 85: What is the purpose of using Requirements Management Tools?

Requirements management tools are used by business analysts to record, manage and prioritize the requirements of a system. They can also be used to manage changes to requirements - something that can be a significant problem for testers as test cases are designed and executed to establish whether the delivered system meets its requirements. Therefore if requirements change after tests have been written then test cases may also need to change. There is also a potential problem of changes not being communicated to all interested parties, thus testers could be using an old set of requirements whilst new ones are being issued to developers.

The use of a traceability function within a requirements tool (and/or integrated with a test management tool) enables links and references to be made between requirements, functions, test conditions and other testware items. This means that as requirements change, it is easy to identify which other items may need to change.

Some requirements management tools can be integrated with test management tools, whilst some test management tools enable requirements to be input and related to test cases.

Requirements management tools also enable requirements coverage metrics to be calculated easily as traceability enables test cases to be mapped to requirements.

As can be seen, traceability can create a lot of maintenance work, but it does highlight those areas that are undergoing change.

Q. 86: What is the purpose of using Configuration Management Tools?

Configuration management tools are designed primarily for managing: the versions of different software and hardware components that comprise a complete build of the system; and various complete builds of systems that exist for various software platforms over a period of time.

<<<<<< =================== >>>>>>

Q. 87: What are the factors on which benefits from Configuration Management Tools depend?

The amount of benefit to be obtained from using configuration management tools is largely dependent upon:

1) The complexity of system architechture;

2) The number and frequency of builds of the integrated system;

3) How much choice is available to customers - whether internal or external.

For example, a software house selling different versions of a product to many customers who run on a variety of operating systems is likely to find configuration management tools more useful than an internal development department working on a single operating system for a single customer.

<<<<<< =================== >>>>>>

Q. 88: What is the importance of traceability between testware and builds?

The use of configuration management tools allows traceability between testware and builds of an integrated system and versions of subsystems and modules. Traceability is useful for:

1) Identifying the correct version of test procedures to be used;

2) Determining which test procedures and other testware can be reused or need to be updated/maintained;

3) Assisting the debugging process so that a failure found when running a test procedure can be traced back to the appropriate version of a subsystem.

when running a test procedure can be traced back to the appropriate version of a subsystem.

<<<<<< =================== >>>>>>

Q. 89: What is the purpose of using review tools in static testing?

Review tools (also known as review process support tools) provide a framework for reviews or inspections. This can include:

1) Maintaining information about the review process, such as rules and checklists.

2) The ability to record, communicate and retain review comments and defects.

3) The ability to amend and reissue the deliverable under review whilst retaining a history or log of the changes made.

4) Traceability functions to enable changes to deliverables under review to highlight other deliverables that may be affected by the change.

5) The use of web technology to provide access from any geographical location to this information.

Review tools can interface with configuration management tools to control the version numbers of a document under review.

If reviews and inspections are already performed effectively then a review tool can be implemented fairly quickly and relatively cheaply. However, if such a tool is used as a means for imposing the use of reviews then the training and implementation costs will be fairly high (as would be the case for implementing a review process without such tools). These tools support the review process, but management buy-in to reviews is necessary if benefits from them are to be obtained in the long run.

Review tools tend to be more beneficial for peer (or technical) reviews and inspections rather than walkthroughs and informal reviews.

<<<<<< =================== >>>>>>

Q. 90: What is the purpose of using Static Analysis Tools?

Static analysis tools analyze code before it is executed in order to identify defects as early as possible. Hence they are used mainly by developers prior to unit testing. A static analysis tool generates lots of error and warning messages about the code.

Training may be required in order to interpret these messages and it may also be necessary to configure the tool to filter out particular types of warning messages that are not relevant. The use of static analysis tools on existing or amended code is likely to result in lots of messages concerning programming standards. A way of dealing with this situation should be considered during the selection and implementation process. For instance, it may be agreed that small changes to existing code should not use the static analysis tool whereas medium to large changes to existing code should use the static analysis tool. A rewrite should be considered if the existing code is significantly non-compliant.

Static analysis tools can find defects that are hard to find during dynamic testing. They can also be used to enforce programming standards (including secure coding), improve the understanding of the code and to calculate complexity and other metrics.

Some static analysis tools are integrated with dynamic analysis tools and coverage measurement tools. They are usually language specific, so to test code written in C++ you would need to have a version of a static analysis tool that was specific to C++.

Other static analysis tools come as part of programming languages or only work with particular development platforms. Note that debugging tools and compilers provide some basic functions of a static analysis tool, but they are generally not considered to be test tools.

The types of defect that can be found using a static analysis tool can include:

1) Syntax errors (e.g. spelling or missing punctuation).

2) Variance from programming standards (e.g. too difficult to maintain).

3) Invalid code structures (missing ENDIF statements).

4) The structure of the code means that some modules or sections of code may not be executed. Such unreachable code or invalid code dependencies may point to errors in code structure.

5) Portability (e.g. code compiles on Windows but not on UNIX).

6) Security vulnerabilities.

7) Inconsistent interfaces between components (e.g. XML messages produced by component A are not of the correct format to be read by component B).

8) References to variables that have a null value or variables declared but never used.

Q. 91: What is the purpose of using Modeling Tools?

Modeling tools are used primarily by developers during the analysis and design stages of the development life cycle. The reason modeling tools are included here is because they are very cost -effective at finding defects early in the development life cycle.

Their benefits are similar to those obtained from the use of reviews and inspections, in that modeling tools allow omissions and inconsistencies to be identified and fixed early so that detailed design and programming can begin from a consistent and robust model. This in turn prevents fault multiplication that can occur if developers build from the wrong model.

For instance, a visual modeling tool using UML can be used by designers to build a model of the software specification. The tool can map business processes to the system architecture model, which, in turn, enables programmers and testers to have a better and common understanding of what programs should do and what testing is required.

Similarly, the use of database, state or object models can help to identify what testing is required and can assist in checking whether tests cover all necessary transactions. Integration with test design tools may also enable modeling tools to support the generation of test cases.

<<<<<< =================== >>>>>>

Q. 92: What is the purpose of using Test Design Tools?

Test design tools are used to support the generation and creation of test cases. In order for the tool to generate test cases, a test basis needs to be input and maintained. Hence many test design tools are integrated with other tools that already contain details of the test basis such as:

1) Modeling tools;

2) Requirements management tools;

3) Static analysis tools;

4) Test management tools.

The level of automation can vary and depends upon the characteristics of the tool itself and the way in which the test basis is recorded in the tool. For example, some tools allow specifications or requirements to be specified in a formal language. This can allow test cases with inputs and expected results to be generated. Other test design tools allow a GUI model of the test basis to be created and then allow tests to be generated from this model.

Some tools - sometimes known as test frames merely generate a partly filled template from the requirement specification held in narrative form. The tester will then need to add to the template and copy and edit as necessary to create the test cases required.

Tests designed from database, object or state models held in modeling tools can be used to verify that the model has been built correctly and can be used to derive some test cases. Tests derived can be very thorough and give high levels of coverage in certain areas.

Some static analysis tools integrate with tools that generate test cases from an analysis of the code. These can include test-input values and expected results.

A test oracle is a type of test design tool that automatically generates expected results. However, these are rarely available as they perform the same function as the software under test.

Test oracles tend to be most useful for:

a) Replacement systems

b) Migrations

c) Regression testing

<<<<<< =================== >>>>>>

Q. 93: What is the purpose of using Test Data Preparation Tools?

Test data preparation tools are used by testers and developers to manipulate data so that the environment is in the appropriate state for the test to be run.

This can involve making changes to the field values in databases, data files, etc., and populating files with a spread of data, (including depersonalized dates of birth, names and addresses, etc. to support data anonymity).

<<<<<< =================== >>>>>>

Q. 94: What is the purpose of using Test Execution Tools?

Test execution tools allow test scripts to be run automatically or at least semi-automatically. A test script (written in a programming language or scripting language) is used to navigate through the system under test and to compare predefined expected outcomes with actual outcomes.

The results of the test run are written to a test log. Test scripts can then be amended and reused to run other or additional scenarios through the same system. Some tools offer GUI-based utilities that enable amendments to be made to scripts more easily than by changing code. These utilities may include:

1) Configuring the script to identify particular GUI objects;

2) Customizing the script to allow it to take specified actions when encountering particular GUI objects or messages;

3) Parameterising the script to read data from various sources.

<<<<<< =================== >>>>>>

Q. 95: What is the purpose of using Recording or Capture Playback Tools?

Record (or capture playback) tools can be used to record a test script and then play it back exactly as it was executed. However, a test script usually fails when played back owing to unexpected results or unrecognized objects. This may sound surprising but consider entering a new customer record onto a system:

1) When the script was recorded, the customer record did not exist. When the script is played

back the system correctly recognizes that this customer record already exists and produces a different response, thus causing the test script to fail.

2) When a test script is played back and actual and expected results are compared a date or time may be displayed. The comparison facility will spot this difference and report a failure.

3) Other problems include the inability of test execution tools to recognize some types of GUI control or object. This might be able to be resolved by coding or reconfiguring the object characteristics (but this can be quite complicated and should be left to experts in the tool).

The expected results are not necessarily captured when recording user actions and therefore may not be compared during playback.

The recording of tests can be useful during exploratory testing for reproducing a defect or for documenting how to execute a test. In addition, such tools can be used to capture user actions so that the navigation through a system can be recorded. In both cases, the script can then be made more robust by a technical expert so that it handles valid system behaviors depending upon the inputs and the state of the system under test.

Q. 96: What is Data-driven testing?

Robust test scripts that deal with various inputs can be converted into data-driven tests. This is where hard-coded inputs in the test script are replaced with variables that point to data in a data-table. Data-tables are usually spreadsheets with one test case per row, with each row containing test inputs and expected results.

The test script reads the appropriate data value from the data-table and inserts it at the appropriate point in the script (e.g. the value in the Customer Name column is inserted into the Customer Name field on the input screen).

<<<<<< =================== >>>>>>

Q. 97: What is Keyword driven testing?

A further enhancement to data-driven testing is the use of keyword-driven (or action word) testing.

Keywords are included as extra columns in the data-table. The script reads the keyword and takes the appropriate actions and subsequent path through the system under test.

Conditional programming constructs such as IF ELSE statements or SELECT CASE statements are required in the test script for keyword-driven testing.

<<<<<< =================== >>>>>>

Q. 98: What are the benefits of using test execution tools?

The efficiency and effectiveness benefits that come from the use of a test execution tool take a long time to come to fruition. First, the selection and implementation process needs to be planned and conducted effectively.

The long-term benefits of test execution tools include:

1) Cost savings as a result of the time saved by running automated tests rather than manual tests.

2) Accuracy benefits from avoiding manual errors in execution and comparison.

3) The ability and flexibility to use skilled testers on more useful and interesting tasks (than running repetitive manual tests).

4) The speed with which the results of the regression pack can be obtained.

5) Note that benefits come primarily from running the same or very similar tests a number of times on a stable platform. Therefore they are generally most useful for regression testing.

<<<<<< =================== >>>>>>

Q. 99: What is the purpose of using Test Harnesses?

Test harnesses are also known as unit test framework tools.

These are used primarily by developers to simulate a small section of the environment in which the software will operate.

They are usually written in-house by developers to perform component or integration testing for a specific purpose.

Test harnesses often use 'mock objects' known as 'stubs' (which stub out the need to have other components or systems by returning predefined values) and 'drivers' (which replace the calling component or system and drive transactions, messages and commands through the software under test).

<<<<<< =================== >>>>>>

Q. 100: What are the benefits of using a test harness to generate XML messages produced by the mainframe?

There are several benefits that can be obtained from using a test harness that iexplore.exe&#0; XML messages produced by the mainframe:

1) It would take a lot of time and effort to design and execute test cases on the mainframe system and run the batch.

2) It would be costly to build a full environment.

3) The mainframe code needed to generate the XML messages may not yet be available.

4) A smaller environment is easier to control and manage. It enables developers (or testers) to perform component and integration testing more quickly as it is easier to localize defects. This allows a quicker turnaround time for fixing defects.

Q. 101: What is the purpose of using Coverage Measurement Tools?

Coverage measurement tools measure the percentage of the code structure covered across white-box measurement techniques such as statement coverage and branch or decision coverage. In addition, they can also be used to measure coverage of modules and function calls. Coverage measurement tools are often integrated with static and dynamic analysis tools and are primarily used by developers.

Coverage measurement tools can measure code written in several programming languages, but not all tools can measure code written in all languages. They are useful for reporting coverage measurement and can therefore be used to assess test completion criteria and/or exit criteria.

Coverage measurement of requirements and test cases/scripts run can usually be obtained from test management tools. This function is unlikely to be provided by coverage measurement tools.

Coverage measurement can be carried out using intrusive or non-intrusive methods. Non-intrusive methods typically involve reviewing code and running code. Intrusive methods, such as 'instrumenting the code' involve adding extra statements into the code. The code is then executed and the extra statements write back to a log in order to identify which statements and branches have been executed.

Instrumentation code can then be removed before it goes into production.

Intrusive methods can affect the accuracy of a test because, for example, slightly more processing will be required to cope with the additional code. This is known as the probe effect and testers need to be aware of its consequences and try to keep its impact to a minimum.

<<<<<< =================== >>>>>>

Q. 102: What is the purpose of using Security Testing Tools?

Security testing tools are used to test the functions that detect security threats and to evaluate the security characteristics of software. The security-testing tool would typically be used to assess the ability of the software under test to:

1) Handle computer viruses;

2) Protect data confidentiality and data integrity;

3) Prevent unauthorized access;

4) Carry out authentication checks of users;

5) Remain available under a denial of service attack;

6) Check non-repudiation attributes of digital signatures.

Security testing tools are mainly used to test e-commerce, e-business and websites. For example, a third-party security application such as a firewall may be integrated into a web-based application.

The skills required to develop and use security tools are very specialized and such tools are generally developed and used on a particular technology platform for a particular purpose. Therefore it maybe worth considering the use of specialist consultancies to perform such testing.

Security tools need to be constantly updated, as there are problems solved and new vulnerabilities discovered all the time - consider the number of Windows XP security releases to see the scale of security problems.

<<<<<< =================== >>>>>>

Q. 103: What is the purpose of using Dynamic Analysis Tools?

Dynamic analysis tools are used to detect the type of defects that are difficult to find during static testing. They are typically used by developers, during component testing and component integration testing, to:

1) Report on the state of the software during its execution;

2) Monitor the allocation, use and de-allocation of memory;

3) Identify memory leaks;

4) Detect time dependencies;

5) Identify unassigned pointers;

6) Check pointer arithmetic.

They are generally used for safety-critical and other high-risk software where reliability is critical.

Dynamic analysis tools are often integrated with static analysis and coverage measurement tools. For example, a developer may want to perform static analysis on code to localize defects so that they can be removed before component test execution.

The integrated tool may allow:

a) The code to be analyzed statically;

b) The code to be instrumented;

c) The code to be executed (dynamically).

Dynamic analysis tools are usually language specific, so to test code written in C++ you would need to have a version of a dynamic analysis tool that was specific to C++.

The tool could then:

a) Report static defects;

b) Report dynamic defects;

c) Provide coverage measurement figures;

d) Report upon the code being (dynamically) executed at various instrumentation points.

<<<<<< =================== >>>>>>

Q. 104: What is the purpose of using Performance Testing/Load Testing/Stress Testing Tools?

Performance testing is very difficult to do accurately and in a repeatable way without using test tools. Therefore performance-testing tools have been developed to carry out both load testing and stress testing.

Load testing reports upon the performance of a system under test, under various loads and usage patterns. A load generator (which is a type of test driver) can be used to simulate the load and required usage pattern by creating virtual users that execute predefined scripts across one or more test machines. Alternatively, response times or transaction times can be measured under various levels of usage by running automated repetitive test scripts via the user interface of the system under test. In both cases output will be written to a log. Reports or graphs can be generated from the contents of the log to monitor the level of performance.

Performance testing tools can also be used for stress testing. In this case, the load on the system under test is increased gradually (ramped up) in order to identify the usage pattern or load at which the system under test fails. For example, if an air traffic control system supports 200 concurrent aircraft in the defined air space, the entry of the 201st or 205th aircraft should not cause the whole system to fail.

Performance testing tools can be used against whole systems but they can also be used during system integration test to test an integrated group of systems, one or more servers, one or more databases or a whole environment.

If the risk analysis finds that the likelihood of performance degradation is low then it is likely that no performance testing will be carried out. For instance, a small enhancement to an existing mainframe system would not necessarily require any formal performance testing. Normal manual testing may be considered sufficient (during which poor performance might be noticed).

There are similarities between performance testing tools and test execution tools in that they both use test scripts and data-driven testing. They can both be left to run unattended overnight and both need a heavy up-front investment, which will take some period of time to pay back.

<<<<<< =================== >>>>>>

Q. 105: What type of defects can be detected by the use of Performance testing tools?

Performance testing tools can find defects such as:

1) General performance problems.

2) Performance bottlenecks.

3) Memory leakage (e.g. if the system is left running under heavy load for some time).

4) Record-locking problems.

5) Concurrency problems.

6) Excess usage of system resources.

7) Exhaustion of disk space.

The cost of some performance tools is high and the implementation and training costs are also high. In addition, finding experts in performance testing is not that easy. Therefore it is worth considering using specialist consultants to come in and carry out performance testing using such tools.

Q. 106: What is the purpose of using Monitoring Tools?

Monitoring tools are used to check whether whole systems or specific system resources are available and whether their performance is acceptable.

Such tools are mainly used in live rather than test environments and are therefore not really testing tools. They tend to be used for monitoring e-commerce, e-business or websites as such systems are more likely to be affected by factors external to the organization and the consequences can be severe in terms of business lost and bad publicity. Generally, if a website is not available, customers will not report it but will go elsewhere.

The use of monitoring tools is generally less important for internal systems as failure is more likely to be noticed only within the organization and contingency plans may also exist. However, the availability of monitoring tools on mainframes, servers and other forms of hardware, means that it is relatively easy to monitor the majority of an organization's infrastructure.

A monitoring tool may be beneficial to monitor the website. A monitoring tool may also exist as part of the mainframe system. However, it is less likely that monitoring tools will be used for the GUI front-end that is used by internal staff.

<<<<<< =================== >>>>>>

Q. 107: What is the purpose of using Data Quality Assessment Tools?

Data quality assessment tools allow files and databases to be compared against a format that is specified in advance. They are typically used on large-scale data intensive projects such as:

1) Conversion of data used on one system into a format suitable for another system.

2) Migration of data from one system to another.

3) Loading of data into a data warehouse.

Data Quality Assessment tools are not specifically designed for testing purposes. They are used primarily for the migration of production data, but typically the development and testing of a migration project will also use these tools.

The data quality assessment tools could be configured to establish whether the customer data being migrated meets particular quality requirements. These requirements may include:

a) Valid postcodes;

b) Valid title for gender;

c) Numeric values in financial fields;

d) Numeric values in date fields.

The tool could also be used to reconcile file record counts with data held in header and footer records to confirm that the number of records in the file equals the number of records loaded into the database.

<<<<<< =================== >>>>>>

Q. 108: What is the purpose of using Usability Test Tools?

Usability test tools typically record the mouse clicks made by remote usability testers when carrying out a specified task.

Some tools also enable other data to be captured such as screenshots, written comments and voice recordings of verbal comments. This recorded data is generally stored in a database so that it can then be analyzed easily by staff at the organization commissioning the testing.

<<<<<< =================== >>>>>>

Q. 109: How do you define a risk while talking about testing?

When we talk about test management we need to talk about associated risk and how it affects the fundamental test process. If there were no risk of adverse future events in software or hardware development then there would be no need for testing. In other words, if defects did not exist then neither would testing.

Risk can be defined as the chance of an event, hazard, threat or situation occurring and its undesirable consequences:

In other words, risk is can be defined as a factor that could result in future negative consequences, usually expressed as impact and likelihood.

<<<<<< =================== >>>>>>

Q. 110: How many broad types of risks are there & how can we define the level of a risk?

In a project a test leader will use risk in two different ways:

1) Project risks

2) Product risks

In both type of risks the calculation of the risk will be as per the following formula

Level of risk = (Probability of the risk occurring) × (Impact if it did happen)

Q. 111: What are the different types of Project Risks?

While managing the testing project a test leader will use project risks to manage the capability to deliver.

Project risks include:

1) Supplier issues:

a) Failure of a third party to deliver on time or at all.

b) Contractual issues, such as meeting acceptance criteria.

c) Organizational factors:

d) Skills, training and staff shortages.

2) Personal issues.

a) Political issues, such as problems that stop testers communicating their needs and test results.

b) Failure by the team to follow up on information found in testing and reviews (e.g. not improving development and testing practices).

c) Improper attitude toward or expectations of testing (e.g. not appreciating the value of finding defects during testing).

3) Technical issues:

a) Problems in defining the right requirements.

b) The extent that requirements can be met given existing project constraints.

c) Test environment not ready on time.

d) Late data conversion, migration planning and development and testing data conversion/migration tools.

e) Low quality of the design, code, configuration data, test data and tests.

For each risk identified a probability (chance of the risk being realized) and impact (what will happen if the risk is realized) should be identified as well as the identification and management of any mitigating actions (actions aimed at reducing the probability of a risk occurring, or reducing the impact of the risk if it did occur).

When analyzing, managing and mitigating these risks the test manager is following well-established project management principles provided within project management methods and approaches. The project risks recognized during test planning should be documented in the IEEE 829 test plan for the ongoing management and control of existing and new project risks a risk register should be maintained by the test leader.

<<<<<< =================== >>>>>>

Q. 112: What are the different types of Product Risks?

When planning and defining tests a test leader or tester using a risk-based testing approach will be managing product risks.

Potential failure areas (adverse future events or hazards) in software are known as product risks, as they are a risk to the quality of the product. In other words, the potential of a defect occurring in the live environment is a product risk. Examples of product risks are:

1) Failure-prone software delivered.

2) The potential that a defect in the software/hardware could cause harm to an individual or company.

3) Poor software characteristics (e.g. functionality, security, reliability, usability, performance).

4) Poor data integrity and quality (e.g. data migration issues, data conversion problems, data transport problems, violation of data standards).

5) Software that does not perform its intended functions.

Risks are used to decide where to start testing in the software development life cycle, e.g. the risk of poor requirements could be mitigated by the use of formal reviews as soon as the requirements have been documented at the start of a project. Product risks also provide information enabling decisions regarding how much testing should be carried out on specific components or systems, e.g. the more risk there is, the more detailed and comprehensive the testing may be. In these ways testing is used to reduce the risk of an adverse effect (defect) occurring or being missed.

Mitigating product risks may also involve non-test activities. For example, in the poor requirements situation, a better and more efficient solution may be simply to replace the analyst who is writing the poor requirements in the first place.

<<<<<< ==================== >>>>>>

Q. 113: What are the benefits of using a risk-based approach to testing?

Risk-based approach to testing provides proactive opportunities to reduce the levels of product risk starting in the initial stages of a project. It involves the identification of product risks and how they are used to guide the test planning, specification and execution.

In a risk-based approach the risks identified:

a) Will determine the test techniques to be employed, and/or the extent of testing to be carried out, e.g. some companies define which test techniques should be used for each level of risk: the higher the risk, the higher the coverage required from test techniques;

b) Prioritize testing in an attempt to find the critical defects as early as possible, e.g. by identifying the areas most likely to have defects (the most complex) the testing can be focused on these areas;

c) Will determine any non-test activities that could be employed to reduce risk, e.g. to provide training to inexperienced designers.

<<<<<< ==================== >>>>>>

Q. 114: What are the after effects of using risk management approach in testing?

Risk-based testing draws on the collective knowledge and insights of the project stakeholders, testers, designers, technical architects, business reps and anyone with knowledge of the solution to determine the risks and the levels of testing to address those risks.

To ensure that the chance of a product failure is minimized, risk management activities provide a disciplined approach:

1) To assess continuously what can go wrong (risks). Regular reviews of the existing and looking for any new product risks should occur periodically throughout the life cycle.

2) To determine what risks are important to deal with (probability × impact). As the project progresses, owing to the mitigation activities risks may reduce in importance, or disappear altogether.

3) To implement actions to deal with those risks (mitigating actions).

Testing supports the identification of new risks by continually reviewing risks of the project deliverables throughout the life cycle; it may also help to determine what risks are important to reduce by setting priorities; it may lower uncertainty about risks by, for example, testing a component and verifying that it does not contain any defects; and lastly by running specific tests it may verify other strategies that deal with risks, such as contingency plans.

Testing is a risk control activity that provides feedback about the residual risk in the product by measuring the effectiveness of critical defect removal and by reviewing the effectiveness of contingency plans.

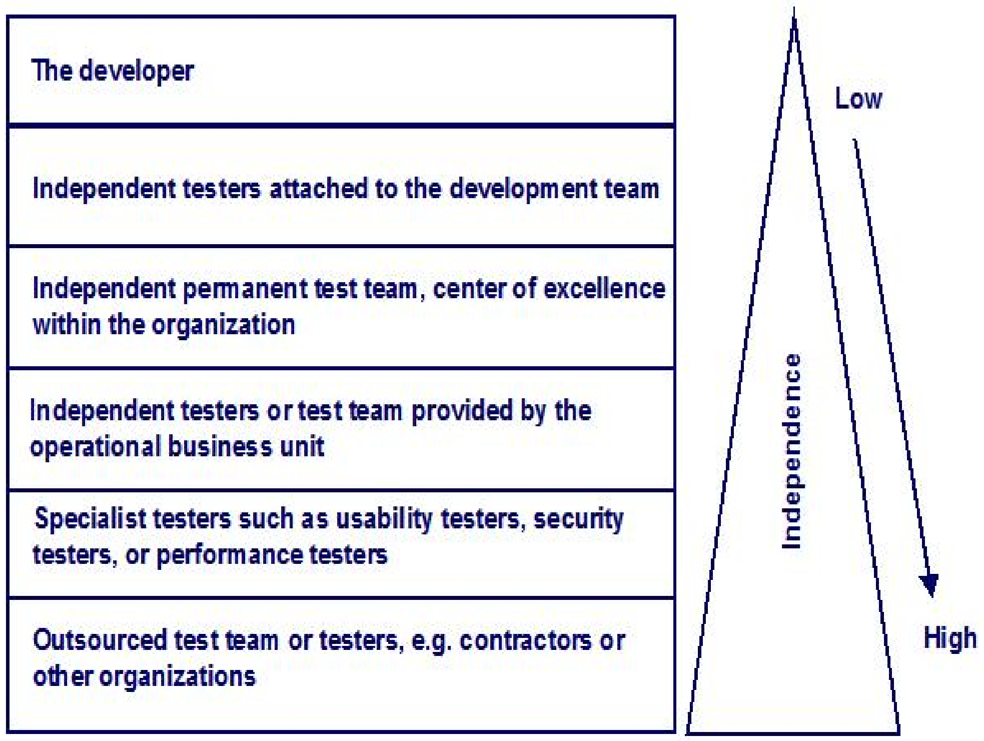
<<<<<< =================== >>>>>>

Q. 115: What are different levels of independence of testing in an organization?

Independent testing is testing carried out by someone other than the creator (developer) of the code being tested. By remaining independent it is possible to improve the effectiveness of testing if implemented correctly.

As humans we are all capable of making mistakes, from the simplest misspelling or wrong use of syntax to fundamental errors at the core of any documents we write. The problem is that as authors we are less able to see our own errors than someone else, who is less directly associated with the document, would be. This is a problem that is made worse, in the world of software development, by the differing ‘world view’ of testers and developers. A developer, as the creator and owner of documents and code related to development, perceives these deliverables as being correct when they are delivered. The general awareness that we all make mistakes is, at this stage, overridden by the belief that what has been produced is what is required. A tester, by contrast, will take the view that anything delivered for testing is likely to contain errors and will search diligently to identify and locate those errors.

This is where independent testing is important, as it is genuinely hard for authors to identify their own errors, but it is easier for others to see them. There are many options for many levels of independence. In general, the more remote a tester is from the production of the document, the greater is the level of independence. Following figure indicates the most common roles and the levels of independence they bring.


The developer
Independent testers attached to the development team
Independent permanent test team, center of excellence within the organization
Independent testers or test team provided by the operational business unit
Specialist testers such as usability testers, security testers, or performance testers
Outsourced test team or testers, e.g. contractors or other organizations
Independence
Low
High

Levels of independent testing

Of course independence comes at a price. The greater the level of independence, the greater the likelihood of errors in testing arising from unfamiliarity. Levels of independence will also depend on the size of the organization. In smaller organizations where everybody contributes to every activity it is harder to differentiate the role of the tester from any other role, and therefore testers may not be very independent at all. The key in these circumstances is for the testers to have independence of mind, not necessarily to be in an independent (separate) team. In organizations where there are clearly defined roles it is a lot easier for a tester to remain independent.

It is also possible to mix and match the levels of independence, e.g. a test team made up of permanent resources, business unit resources and contractors. For large, complex or safety-critical projects, it is usually best to have multiple levels of testing, with some or all of the levels done by independent testers.

The ‘agile’ approach to development challenges the traditional approach to independence. In this approach everybody takes on multiple roles and so maintaining total independence is not always possible. A tester in this situation has to be able to switch to an independent view, at the relevant points in the project. Testers achieve this independence of view by not assuming anything and by not starting to own the software like a developer would, e.g. the view that was how it was developed to work.

Q. 116: What are the various Pros & Cons of Implementing Independent Testing?

Independence in the implementation of testing has some key benefits and drawbacks, as described in the following table.

Sr.	Benefits	Drawbacks
1	The tester sees other and different defects to the author	Isolation from the development team (if treated as totally independent), which will mean the tester is totally dependent on the test basis to understand what it is the tester is testing (documentation that is rarely up to date)
2	The tester is unbiased	The tester may be seen as the bottleneck, as independent test execution is normally the last stage and affected by any delays earlier in the process
3	The tester can see what has been built rather than what the developer thought had been built	Developers lose a sense of responsibility for quality as it may be assumed that they need not worry about errors because the independent test team will find them
4	The tester makes no assumptions regarding quality	The fully independent view sets developers and testers on either side of an invisible fence. This can be a hindrance to communication that would in normal circumstances ensure common understanding and effective working. It can also mean that developers are seen to 'throw' the software over the fence

<<<<<< =================== >>>>>>

Q. 117: Testing can be done by which type of people?

Testing tasks are traditionally carried out by people who make testing a career; however, testing tasks may also be carried out by non-testers such as a project manager, quality manager, developer, business and domain expert, infrastructure or IT operations.

The availability of resources usually determines the resource types that are deployed on each project, e.g. if there are no career testers available an organization may identify non-testing IT or business resources to carry out the role of tester for a specific project or time period.

The syllabus defines two testing roles, the test leader (or test manager/test coordinator) and the tester.

<<<<<< =================== >>>>>>

Q. 118: what is the difference between a testing role and a testing job?

There is great difference between a testing role and a testing job.

A role is an activity, or a series of activities given to a person to fulfil, e.g. the role of test leader. A person may therefore have more than one role at any moment depending on their experience and the level of workload on a project.

A job is effectively what an individual is employed to do, so one or many roles could make up a job. For example, a test leader could also be a tester.

<<<<<< =================== >>>>>>

Q. 119: What are the tasks undertaken by a test leader?

The tasks undertaken by a test leader align very closely with those undertaken by a project manager and align closely with standard approaches to project management. In this context a test leader is anyone who leads a team of testers. They are also known as test program managers, test managers, test team leaders and test coordinators.

Typical test leader tasks may include:

1) Coordinating the development of the test strategy and plan with project managers and others.

2) Writing or reviewing test strategies produced for the project, and test policies produced for the organization.

3) Contributing the testing perspective to other project activities, such as development delivery schedules.

4) Planning the development of the required tests - which will include ensuring that the development uses the correct understanding of the risks, selecting the required test approaches (test levels, cycles, approach, objectives and incident management planning), estimating the time and effort and converting to the cost of testing and acquiring the right resources.

5) Managing the specification, preparation, implementation and execution of tests, including the monitoring and control of all the specification and execution.

6) Taking the required action, including adapting the planning, based on test results and progress (sometimes documented in status reports), and any action necessary to compensate for problems or delays.

7) Ensuring that adequate configuration management of testware is in place and that the testware is fully traceable, e.g. there is a hierarchical relationship established between the requirements and the detailed specification documents.

8) Putting in place suitable metrics for measuring test progress and evaluating the quality of the testing delivered and the product.

9) Agreeing what should be automated, to what degree, and how, ensuring it is implemented as planned.

10) Where required, selecting tools to support testing and ensuring any tool training requirements

are met.

11) Agreeing the structure and implementation of the test environment.

12) Scheduling all testing activity.

13) At the end of the project, writing a test summary report based on the information gathered during testing.

<<<<<< =================== >>>>>>

Q. 120: What are the tasks undertaken by a tester?

Generally a tester is also known as test analyst or test executor.

The tasks typically undertaken by a tester may include:

1) Reviewing and contributing to the development of test plans.

2) Analyzing, reviewing and assessing user requirements, specifications and models for testability.

3) Creating test specifications from the test basis.

4) Setting up the test environment (often coordinating with system administration and network management). In some organizations the setting up and management of the test environment could be centrally controlled; in this situation a tester would directly do liaison with the environment management to ensure the test environment is delivered on time and to specification.

5) Preparing and acquiring/copying/creating test data.

6) Implementing tests on all test levels, executing and logging the tests, evaluating the results and documenting the deviations from expected results as defects.

7) Using test administration or management and test monitoring tools as required.

8) Automating tests that may be supported by a developer or a test automation expert.

9) Where required, running the tests and measuring the performance of components and systems.

10) Reviewing tests developed by other testers.

Q. 121: What is a test approach & what is the right time to formulate the test approach?

A test approach is the implementation of a test strategy on a particular project. The test approach defines how testing will be implemented. A test approach can reflect testing for a whole organization, a program of work or an individual project.

Test approach can be:

1) Developed early in the life cycle, which is known as preventative - in this approach the test design process is initiated as early as possible in the life cycle to stop defects being built into the final solution;

2) Deferred until just before the start of test execution, which is known as reactive - this is where testing is the last development stage and is not started until after design and coding has been completed (sometimes it is identified as the waterfall approach, i.e. all development stages are sequential, the next not starting until the previous one has nearly finished).

A test approach includes all of the decisions made on how testing should be implemented, based upon the (test) project goals and objectives, as well as the risk assessment. It forms the starting point for test planning, selecting the test design techniques and test types to be employed. It should also define the software and test entry and exit criteria.

<<<<<< =================== >>>>>>

Q. 122: What are the different approaches or strategies that can be deployed in Testing projects?

There are many approaches or strategies that can be employed. All will depend on the context within which the test team is working, and may consider risks, hazards and safety, available resources and skills, the technology, the nature of the system (e.g. custom built vs COTS), test objectives and regulations, and may include:

1) Analytical approaches: Such as risk-based testing where testing is directed to areas of greatest risk (see later in this section for an overview of risk-based testing).

2) Model-based approaches: Such as stochastic testing using statistical information about failure rates (such as reliability growth models) or usage (such as operational profiles).

3) Methodical approaches: Such as failure-based (including error guessing and fault attacks), checklist based and quality-characteristic based.

4) Standard-compliant approaches: As specified by industry-specific standards such as The Railway Signaling standards (which define the levels of testing required) or the MISRA (which defines how to design, build and test reliable software for the motor industry).

5) Process-compliant approaches: These adhere to the processes developed for use with the various agile methodologies or traditional waterfall approaches.

6) Dynamic and heuristic approaches: Such as exploratory testing where testing is more reactive to events than pre-planned, and where execution and evaluation are concurrent tasks.

7) Consultative approaches: Such as those where test coverage is driven primarily by the advice and guidance of technology and/or business domain experts outside or within the test team.

8) Regression-averse approaches: Such as those that include reuse of existing test material, extensive automation of functional regression tests, and standard test suites.

Different approaches may be combined if required. The decision as to how and why they will be combined will depend on the circumstances prevalent in a project at the time. For example, an organization may as a standard use an agile method, but in a particular situation the structure of the test effort could use a risk-based approach to ensure the testing is correctly focused.

<<<<<< =================== >>>>>>

Q. 123: What are the factors that need to be considered when defining the strategy or approach for testing?

Deciding on which approach to take may be dictated by standards, e.g. those used in the motor industry that is set by MISRA, or at the discretion of those developing the approach or strategy. Where discretion is allowed, the context or scenario needs to be taken into account.

Following factors are considered when defining the strategy or approach:

1) Risk of failure of the project, hazards to the product and risks of product failure to humans, the environment and the company, e.g. the cost of failure would be too high (safety-critical environments).

2) Skills and experience of the people in the proposed techniques, tools and methods. There is no point in implementing a sophisticated component-level, technique-driven approach or strategy when the only resources available are business users with no technical grounding.

3) The objective of the testing endeavor and the mission of the testing team, e.g. if the objective is to find only the most serious defects.

4) Regulatory aspects, such as external and internal regulations for the development process, e.g. The Railway Signaling standards that enforce a given level of test quality.

5) The nature of the product and the business, e.g. a different approach is required for testing mobile phone coverage than for testing an online banking operation.

<<<<<< =================== >>>>>>

Q. 124: What do we mean by Test Planning?

Test planning is the most important activity undertaken by a test leader in any test project. It ensures that there is initially a list of tasks and milestones in a baseline plan to track progress against, as well as defining the shape and size of the test effort. Test planning is used in development and implementation projects as well as maintenance activities involving changes and fixes.

The main document produced in test planning is often called a master test plan or a project test plan. This document defines the high level of the test activities being planned. It is normally produced during the early phases of the project (e.g. initiation) and will provide sufficient information to enable a test project to be established (bearing in mind that at this point in a project little more than requirements may be available from which to plan).

<<<<<< =================== >>>>>>

Q. 125: What are the different sections of a Test plan?

The details of the test-level activities are documented within test-level plans, e.g. the system test plan.

These documents will contain the detailed activities and estimates for the relevant test level.

The IEEE 829 standard identifies that there should be a minimum of 16 sections present in a test plan, as described in the following table.

Section No.	**Heading**	**Details**
1	Test plan identifier	A unique identifying reference such as Doc ref XYZ v2?
2	Introduction	A brief introduction to the document and the project for which it has been produced
3	Test items	# A test item is a software item that is the object of testing # A software item is one or more items of source code, object code, job control code, or control data # This section should contain any documentation references, e.g. design documents
4	Features to be tested	# A feature is a distinguishing characteristic of a software item (e.g. performance, portability, or functionality) # Identify all software features and combinations of features and the associated test design specification
5	Features not to be tested	Identify all software features and significant combinations and state the reasons for not including them
6	Approach	Details the overall approach to testing; this could include a detailed process definition, or could refer to other documentation where the detail is documented, i.e. a test strategy
7	Item pass / fail	Used to determine whether a software item has

	criteria	passed or failed its test
8	Suspension and resumption requirements	Suspension requirements define criteria for stopping part or all of the testing activity Resumption requirements specify the requirements to resume testing
9	Test deliverables	The documents that testing will deliver, e.g. from IEEE 829 documents such as: # Test plans (for each test level) # Test specifications (design, case and procedure) # Test summary reports
10	Testing tasks	All tasks for planning and executing the testing, including the inter-task dependencies
11	Environmental needs	Definition of all environmental requirements such as hardware, software, PCs, desks, stationery, etc.
12	Responsibilities	Identifies the roles and tasks to be used in the test project and who will own them
13	Staffing and training needs	Identifies any actual staffing requirements and any specific skills and training requirements, e.g. automation
14	Schedule	Document delivery dates and key milestones
15	Risks and contingencies	High-level project risks and assumptions and a contingency plan for each risk
16	Approvals	Identifies all approvers of the document, their titles and the date of signature

Q. 126: What is the short cut method to remember all 16 sections of a test plan as described in IEEE 829?

A useful revision aid to help remember the 16 sections of the IEEE 829 test plan is the acronym ‘SPACEDIRT’. Here each letter refers to one or many sections of the test plan as described below:

S - Scope (including test items, features to be tested and features not to be tested)

P - People (including responsibilities, staff and training and approvals)

A - Approach

C - Criteria (including item pass/fail criteria and suspension and resumption requirements)

E - Environment needs

D - Deliverables (test)

I - Identifier and introduction (test plan)

R - Risks and contingencies

T - Testing tasks and schedule

<<<<<< =================== >>>>>>

Q. 127: What are the different Activities involved in Test-Planning?

During test planning various activities for an entire system or a part of a system have to be undertaken by those working on the plan. These activities are:

1) Working with the project manager and subject matter experts' to determine the scope and the risks that need to be tested. As well identifying and agreeing the objectives of the testing, be they time, quality or cost focussed, or in fact maybe a mixture of all three. The objectives will enable the test project to know when it has finished.

2) Putting together the overall approach of testing (sometimes called the test strategy), ensuring that the test levels and entry and exit criteria are defined.

3) Liaison with the project manager and making sure that the testing activities have been included within the software life-cycle activities such as:

a) Design - The development of the software design;

b) Development – The building of the code;

c) Implementation - The activities surrounding implementation into a live environment.

4) Working with the project to decide what needs to be tested, what roles are involved and who

will perform the test activities, planning when and how the test activities should be done, deciding how the test results will be evaluated, and defining when to stop testing (exit criteria).

5) Building a plan that identifies when and who will undertake the test analysis and design activities. In addition to the analysis and design activities test planning should also document the schedule for test implementation, execution and evaluation.

6) Finding and assigning resources for the different activities that have been defined.

7) Deciding what the documentation for the test project will be, e.g. which plans, how the test cases will be documented, etc.

8) Defining the management information, including the metrics required and putting in place the processes to monitor and control test preparation and execution, defect resolution and risk issues.

9) Ensuring that the test documentation generates repeatable test assets, e.g. test cases.

<<<<<< =================== >>>>>>

Q. 128: What are the different Entry Criteria for the test execution?

Entry criteria are used to determine when a given test activity can start. This could include the beginning of a level of testing, when test design and/or when test execution is ready to start.

Examples of some typical entry criteria to test execution include:

1) Test environment available and ready for use (it functions).

2) Test tools installed in the environment are ready for use.

3) Testable code is available.

4) All test data is available and correct.

5) All test design activity has completed.

<<<<<< =================== >>>>>>

Q. 129: What are the different Exit Criteria for the test execution?

Exit criteria are used to determine when a given test activity has been completed or when it should stop. Exit criteria can be defined for all of the test activities, such as planning, specification and execution as a whole, or to a specific test level for test specification as well as execution.

Exit criteria should be included in the relevant test plans.

Some typical exit criteria are:

1) All tests planned have been run.

2) A certain level of requirements coverage has been achieved.

3) No high-priority or severe defects are left outstanding.

4) All high-risk areas have been fully tested, with only minor residual risks left outstanding.

5) Cost—when the budget has been spent.

The schedule has been achieved, e.g. the release date has been reached and the product has to go live.

Exit criteria should have been agreed as early as possible in the life cycle; however, they can be and often are subject to controlled change as the detail of the project becomes better understood and therefore the ability to meet the criteria is better understood by those responsible for delivery.

<<<<<< =================== >>>>>>

Q. 130: What is Metrics-Based Approach of Test Estimation?

Metrics based approach relies upon data collected from previous or similar projects.

This kind of data can include the following:

1) The number of test conditions.

2) The number of test cases written.

3) The number of test cases executed.

4) The time taken to develop test cases.

5) The time taken to run test cases.

6) The number of defects found.

7) The number of environment outages and how long on average each one lasted.

With this approach and data it is possible to estimate quite accurately what the cost and time required for a similar project would be.

It is important that the actual costs and time for testing are accurately recorded. These can then be used to revalidate and possibly update the metrics for use on the next similar project.

Q. 131: What is Expert-Based Approach of Test Estimation?

This approach uses the experience of owners of the relevant tasks or experts to derive an estimate (this is also known as the Wide Band Delphi approach).

In this context 'experts' could be:

1) Business experts.

2) Test process consultants.
3) Developers.
4) Technical architects.
5) Analysts and designers.
6) Anyone with knowledge of the application to be tested or the tasks involved in the process.

There are many ways that this approach could be used. Following are the two examples:

1) Distribute a requirement specification to the task owners and get them to estimate their task in isolation. Amalgamate the individual estimates when received; build in any required contingency, to arrive at the estimate.

2) Distribute to known experts who develop their individual view of the overall estimate and then meet together to agree on and/or debate the estimate that will go forward.

Expert estimating can use either of the above approaches individually or mixing and matching them as required.

<<<<<< =================== >>>>>>

Q. 132: What are the categories of levels of effort required to fulfil the test requirements in a project?

Many things affect the level of effort required to fulfil the test requirements of a project.

These can be split into three main categories, as described below.

1) Product characteristics:

a) Size of the test basis;
b) Complexity of the final product;
c) The amount of non-functional requirements;
d) The security requirements (perhaps meeting BS 7799, the security standard);
e) How much documentation is required (e.g. some legislation-driven changes demand a certain level of documentation that may be more than an organization would normally produce);
f) The availability and quality of the test basis (e.g. requirements and specifications).

2) Development process characteristics:

a) Times-scales;

b) Amount of budget available;

c) Skills of those involved in the testing and development activity (the lower the skill level in development, the more defects could be introduced, and the lower the skill level in testing, the more detailed the test documentation needs to be);

d) Which tools are being used across the life cycle (i.e. the amount of automated testing will affect the effort required).

3) Expected outcome of testing such as:

a) The amount of errors;
b) Test cases to be written.

Taking all of this into account, once the estimate is developed and agreed the test leader can set about identifying the required resources and building the detailed plan.

<<<<<< =================== >>>>>>

Q. 133: How do we track & control the progress of testing projects?

Having developed the test plan, the activities and time-scales determined within it need to be constantly reviewed against what is actually happening. This is test progress monitoring. The purpose of test progress monitoring is to provide feedback and visibility of the progress of test activities.

The data required to monitor progress can be collected manually, e.g. counting test cases developed at the end of each day, or, with the advent of sophisticated test management tools, it also possible to collect the data as an automatic output from a tool either already formatted into a report, or as a data file that can be manipulated to present a picture of progress.

The progress data is also used to measure exit criteria such as test coverage, e.g. 50 per cent requirements coverage achieved.

Common test metrics include:

1) Percentage of work done in test case preparation (or percentage of planned test cases prepared).

2) Percentage of work done in test environment preparation.

3) Test case execution (e.g. number of test cases run/not run, and test cases passed/failed).

4) Defect information (e.g. defect density, defects found and fixed, failure rate and retest results).

5) Test coverage of requirements, risks or code.

6) Subjective confidence of testers in the product.

7) Dates of test milestones.

8) Testing costs, including the cost compared with the benefit of finding the next defect or to run the next test.

Ultimately test metrics are used to track progress towards the completion of testing, which is determined by the exit criteria. So test metrics should relate directly to the exit criteria.

<<<<<< =================== >>>>>>

Q. 134: What information should be included while Test Reporting?

Test reporting is the process where by test metrics are reported in summarized format to update the reader regarding the testing tasks undertaken. The information reported can include:

1) What has happened during a given period of time, e.g. a week, a test level or the whole test endeavor, or when exit criteria have been met.

2) Analyzed information and metrics required to support recommendations and decisions about future actions, such as:

a) An assessment of defects remaining;

b) The economic benefit of continued testing, e.g. additional tests are exponentially more expensive than the benefit of running;

c) Outstanding risks;

d) The level of confidence in tested software, e.g. defects planned vs. actual defects found.

<<<<<< =================== >>>>>>

Q. 135: What are the different sections of a Test Summary Report?

The IEEE 829 standard includes an outline of a test summary report that could be used for test reporting. The structure defined in the outline is shown in the following table.

Section No.	**Heading**	**Details**
1	Test summary report identifier	The specific identifier allocated to this document, e.g. TSR XYX v1
2	Summary	# Identifies the items tested (including any version numbers) # Documents the environments in which the test activity being reported on took place # References the testing documentation for each test item, e.g. test plan, test cases, test defect reports
3	Variances	Reports deviations from the test approach or strategy, test plan, test specification or test procedures
4	Comprehensive assessment	Measures the actual progress made against the exit criteria and explains why any differences have arisen

5	Summary results	Provides an overview of the results from the test activities; it should include details of defects raised and fixed, as well as those that remain unresolved
6	Evaluation	Provides an evaluation of the quality of each test item, including a view of the risks of failure in production of these test items. Based upon the test result metrics and test item pass/fail criteria
7	Summary of activities	# A summary of the major test activities and events such as test environment unavailability, success or weaknesses of the test specification process, etc. # Should also include resource usage data, e.g. planned spend against actual spend
8	Approvals	Identifies all approvers of the document

The information gathered can also be used to help with any process improvement opportunities.

This information could be used to assess whether:

a) The goals for testing were correctly set (were they achievable; if not why not?);

b) The test approach or strategy was adequate (e.g. did it ensure there was enough coverage?);

c) The testing was effective in ensuring that the objectives of testing were met.

Q. 136: What are the different test-control activities?

We have referred above to the collection and reporting of progress data. Test control uses this information to decide on a course of action to ensure control of the test activities is maintained and exit criteria are met. This is particularly required when the planned test activities are behind schedule. The actions taken could impact any of the test activities and may also affect other software life-cycle activities.

Examples of test-control activities are as follows:

1) Making decisions based on information from test monitoring.

2) Re-prioritize tests when an identified project risk occurs (e.g. software delivered late).

3) Change the test schedule due to availability of a test environment.

4) Set an entry criterion requiring fixes to be re-tested (confirmation tested) by a developer before accepting them into a build (this is particularly useful when defect fixes continually fail again when re-tested).

5) Review of product risks and perhaps changing the risk ratings to meet the target.

6) Adjusting the scope of the testing (perhaps the amount of tests to be run) to manage the testing of late change requests.

<<<<<< =================== >>>>>>

Q. 137: Which test-control activities don't fall under the responsibility of test leader?

The following test-control activities are likely to be outside the test leader's responsibility. However, this should not stop the test leader making a recommendation to the project manager.

1) Descoping of functionality, i.e. removing some less important planned deliverables from the initial delivered solution to reduce the time and effort required to achieve that solution.

2) Delaying release into the production environment until exit criteria have been met.

3) Continuing testing after delivery into the production environment so that defects are found before they occur in production.

<<<<<< =================== >>>>>>

Q. 138: What is Incident Management

An incident is any unplanned event occurring that requires further investigation. In testing this translates into anything where the actual result is different to the expected result. An incident when investigated may be a defect, however, it may also be a change to a specification or an issue with the test being run. It is important that a process exists to track all incidents through to closure.

Incidents can be raised at any time throughout the software development life cycle, from reviews of the test basis (requirements, specifications, etc.) to test specification and test execution.

Incident management, according to IEEE 1044 (Standard Classification for Software Anomalies), is 'The process of recognizing, investigating, taking action and disposing of incidents.' It involves recording incidents, classifying them and identifying the impact.

The process of incident management ensures that incidents are tracked from recognition to correction, and finally through retest and closure. It is important that organizations document their incident management process and ensure they have appointed someone (often called an incident manager / coordinator) to manage / police the process.

<<<<<< =================== >>>>>>

Q. 139: What are the general objectives of Incident reports?

Incidents are raised on incident reports, either electronically via an incident management system (from Microsoft Excel to sophisticated incident management tools) or on paper.

Incident reports have the following objectives:

1) To provide developers and other parties with feedback on the problem to enable identification, isolation and correction as necessary. It must be remembered that most developers and other parties who will correct the defect or clear up any confusion will not be present at the point of identification, so without full and concise information they will be unable to understand the problem, and possibly therefore unable to understand how to go about fixing it. The more information provided, the better.

2) To provide test leaders with a means of tracking the quality of the system under test and the progress of the testing. One of the key metrics used to measure progress is a view of how many incidents are raised, their priority and finally that they have been corrected and signed off.

3) To provide ideas for test process improvement. For each incident the point of injection should be documented, e.g. a defect in requirements or code, and subsequent process improvement can focus on that particular area to stop the same defect occurring again.

<<<<<< =================== >>>>>>

Q. 140: What are the general contents of an incident report?

The details that are normally included on an incident report are:

1) Date of issue, issuing organization, author, approvals and status.

2) Scope, severity and priority of the incident.

3) References, including the identity of the test case specification that revealed the problem.

4) Expected and actual results.

5) Date the incident was discovered.

6) Identification of the test item (configuration item) and environment.

7) Software or system life-cycle process in which the incident was observed.

8) Description of the incident to enable reproduction and resolution, including logs, database dumps or screenshots.

9) Degree of impact on stakeholder(s) interests.

10) Severity of the impact on the system.

11) Urgency/priority to fix.

12) Status of the incident (e.g. open, deferred, duplicate, waiting to be fixed, fixed awaiting confirmation test or closed).

13) Conclusions, recommendations and approvals.

14) Global issues, such as other areas that may be affected by a change resulting from the

incident.

15) Change history, such as the sequence of actions taken by project team members with respect to the incident to isolate, repair and confirm it as fixed.

As per IEEE 829 an outline of a test incident report contains different sections shown in following table.

Section No.	**Heading**	**Details**
1	Test incident report identifier	The unique identifier assigned to this test incident report
2	Summary	A summary of the incident, detailing where expected and actual results differ, identifying at a high level the items that are affected, and the steps leading up to the recognition of the incident, e.g. if in test execution, which test was executed and the actual keystrokes and data loaded
3	Incident description	A detailed description of the incident, which should include: # Inputs # Expected results # Actual results # Anomalies # Date and time # Procedure step # Environment # Attempts to repeat # Testers' comments # Observers' comments Should also include any information regarding

		possible causes and solutions
4	Impact	If known, document what impact the incident has on progress

Q. 141: What is Configuration management?

Configuration management is the process of managing products, facilities and processes by managing the information about them, including changes, and ensuring they are what they are supposed to be in every case.

For testing, configuration management will involve controlling both the versions of code to be tested and the documents used during the development process, e.g. requirements, design and plans.

In both instances configuration management should ensure traceability throughout the test process, e.g. a requirement should be traceable through to the test cases that are run to test its levels of quality, and vice versa.

Effective configuration management is important for the test process as the contents of each release of software into a test environment must be understood and at the correct version, otherwise testers could end up wasting time because either they are testing an invalid release of the software or the release does not integrate successfully, leading to the failure of many tests.

In most instances the project will have already established its configuration management processes that will define the documents and code to be held under configuration management. If this is not the case then during test planning the process and tools required to establish the right configuration management processes will need to be selected/implemented by the test leader.

The same principle applies to testware. Each item of testware (such as a test procedure) should have its own version number and be linked to the version of the software it was used to test. For example, test procedure TP123a might be used for Release A and TP123b might be used for Release B - even though both have the same purpose and even expected results. However, another test procedure, TP201, may be applicable to all releases.

A good configuration management system will ensure that the testers can identify exactly what code they are testing, as well as have control over the test documentation such as test plans, test specification, defect logs, etc.

<<<<<< =================== >>>>>>

Q. 142: What is the importance of test independence in a test organization?

The importance of independence in the test organization and how independence helps to ensure that the right focus is given to the test activity was reviewed.

Independence is gained by separating the creative development activity from the test activity and we can have following levels of independence that are achievable:

a) The developers - low independence.

b) Independent testers ceded to the development team.

c) Independent permanent test team, center of excellence with the organization.

d) Independent testers or test team provided by the operational business unit.

e) Specialist testers such as security testers or performance testers.

f) Outsourced test team or the use of independent contractors - high independence.

<<<<<< =================== >>>>>>

Q. 143: What is the importance of configuration management system in a test organization?

When running test cases against the code it is important that the tester is aware of the version of code being tested and the version of the test being run. Controlling the versioning of the software and test assets is called configuration management.

Lack of configuration management may lead to issues like loss of already-delivered functionality, reappearance of previously corrected errors and no understanding of which version of test was run against which version of code.

<<<<<< =================== >>>>>>

Q. 144: What are the main tests planning documents as per IEEE 829?

IEEE 829, the test documentation standard, provides outlines of three test planning documents:

1) The test plan

2) The test summary report

3) The test incident report

Test management depends not only on the preparation of the required documents but also on the development of the right entry and exit criteria and estimates, the monitoring of progress through the plan and the control activities implemented to ensure the plan is achieved.

<<<<<< =================== >>>>>>

Q. 145: What are software compilers?

Software compilers are computer programs (or a set of programs) that translate codes written in one computer language (the source language) into another computer language (the target language). As part of the compile process certain static analysis can be undertaken that will identify some defects and provide the calculation of software metrics.

Q. 146: How do we classify different types of testing tools?

Test tools can be classified according to the type of activity they support. The test tools can be classified according to:

a) Their purpose;

b) The activities within the fundamental test process and the software life cycle with which they are primarily associated;

c) The type of testing that they support;

d) The source of tool (shareware, open source, free or commercial);

e) The technology used;

f) Who uses them.

<<<<<< =================== >>>>>>

Q. 147: How do we use a typical incident management tool?

An incident management tool can be used to raise new defects and process them through the defect life cycle until resolved. It can also be used to check whether defects (or similar defects) have been raised before, especially for defects raised during regression testing.

An incident management tool could also be used to prioritize defects so that developers fix the most important ones first. It could also highlight clusters of defects. This may suggest that more detailed testing needs to be done on the areas of functionality where most defects are being found as it is probable that further defects will be found as well.

<<<<<< =================== >>>>>>

Q. 148: How do you define a build?

A build is a development activity where a complete system is compiled and linked (typically daily) so that a consistent system is available at any time including all the latest changes.

<<<<<< =================== >>>>>>

Q. 149: What are the tests Comparators?

Test Comparators compare the contents of files, databases, XML messages, objects and other electronic data formats. This allows expected results and actual results to be compared. They can also highlight differences and thus provide assistance to developers when localizing and debugging code.

They often have functions that allow specified sections of the file, screen or object to be ignored or masked out. This means that a date or time stamp on a screen or field can be masked out as it is expected to be different when a comparison is performed.

Comparators are particularly useful for regression testing since the contents of output or interface files should usually be the same. This is probably the test tool that provides the single greatest benefit. For instance, manually comparing the contents of a database query containing thousands of rows is time-consuming, error prone and demotivating. The same task can be performed accurately and in a fraction of the time using a Comparators. Comparators are usually included in test execution tools.

<<<<<< =================== >>>>>>

Q. 150: What are the tools that are not meant for testers as such but are extensively used in test organizations?

Other tools that are not designed specifically for testers or developers can also be used to support one or more test activities. These include:

1) Spreadsheets

2) Word processors

3) Email

4) Back-up and restore utilities

5) SQL and other database query tools

6) Project planning tools

7) Debugging tools (although these are more likely to be used by developers than testers).