Software Testing

Time: 3 Hrs.] Prelim Question Paper solution [Marks : 100

Q.1(a) Attempt any THREE of the following : [12]

Q.1(a) (i) List eight traits (properties) of good software tester? [4]

Ans.: (1) Be Skeptical [Any Eight - ½ mark each]

Don't believe that the build given by developers is bug free or quality outcome. Question everything. Accept the build only if you test and find it defect free. Don't believe anyone whatever be the designation they hold, just apply your knowledge and try to find errors. You need to follow this till the last testing cycle.

(2) Don't Compromise on Quality

Don't compromise after certain testing stages. There is no limit for testing until you produce a quality product. Quality is the word made by software testers to achieve more effective testing. Compromising at any level leads to defective product, so don't do that at any situation.

(3) Ensure End User Satisfaction

Always think what can make end user happy. How they can use the product with ease. Don't stop by testing the standard requirements. End user can be happy only when you provide an error free product.

(4) Think from Users Perspective

Every product is developed for customers. Customers may or may not be technical persons. If you don't consider the scenarios from their perspective you will miss many important bugs. So put yourself in their shoes. Know your end users first. Their age, education even the location can matter most while using the product. Make sure to prepare your test scenarios and test data accordingly. After all project is said to be successful only if end user is able to use the application successfully.

(5) Prioritize Tests

First identify important tests and then prioritize execution based on test importance. Never ever execute test cases sequentially without deciding priority. This will ensure all your important test cases get executed early and you won't cut down on these at the last stage of release cycle due to time pressure. Also consider the defect history while estimating test efforts. In most cases defect count at the beginning is more and goes on reducing at the end of the test cycle.

(6) Never Promise 100% Coverage

Saying 100% coverage on paper is easy but practically it is impossible. So never promise to anyone including clients about total test coverage. In business there is a philosophy – "Under promise and over deliver." So don't goal for 100% coverage but focus on quality of your tests.

(7) Be Open to Suggestions

Listen to everyone even though you are an authority on the project having in depth project knowledge. There is always scope for improvements and getting suggestions from fellow software testers is a good idea. Everyone's feedback to improve the quality of the project would certainly help to release a bug free software.

(8) Start Early

Don't wait until you get your first build for testing. Start analyzing requirements, preparing test cases, test plan and test strategy documents in early design phase. Starting early to test helps to visualize complete project scope and hence planning can be done accordingly. Most of the defects can be detected in early design and analysis phase saving huge time and money. Early requirement analysis will also help you to question the design decisions.

(9) Identify and Manage Risks

Risks are associated with every project. Risk management is a three step process. Risk identification, analysis and mitigation. Incorporate risk driven testing process. Priorities software testing based on risk evaluation.

(10) Do Market Research

Don't think that your responsibility is just to validate software against the set of requirements. Be proactive, do your product market research and provide suggestions to improve it. This research will also help you understand your product and its market.

(11) Develop Good Analyzing Skill

This is must for requirement analysis but even further this could be helpful for understanding customer feedback while defining test strategy. Question everything around you. This will trigger the analysis process and it will help you resolve many complex problems.

(12) Focus on Negative Side as Well

Testers should have test to break attitude. Concentrating on only positive side will almost certainly create many security issues in your application. You should be hacker of your project to keep other hackers away from it. Negative testing is equally important. So cover a good chunk of your test cases based on negative scenarios.

(13) Be a Good Judge of Your Product

Judge usually thinks whether it is right or wrong. Judge listens to both the sides. Same is applicable for testing. As a software tester if you think something as right, try to prove it why it is not wrong and then only accept it. You must have valid reason for all your decisions.

(14) Learn to Negotiate

Testers have to negotiate with everyone in all stages of project life cycle. Especially negotiation with developers is more important. Developers can do anything to prove that their code is correct and the defect logged by testers is not valid. It requires great skills to convince developers about the defect and get it resolved. Though some software testers think this is not our task but explaining the true impact of any issue is very helpful for developers to quickly understand the overall scenario and its implications. This requires years of practice but once you learn to negotiate you will gain more respect.

(15) Stop the Blame Game

It's common to blame others for any defects which are not caught in testing. This is even more common when the tester's responsibilities are not defined concretely. But in any situation never blame anyone. If an error occurs, first try to resolve it rather than finding someone to blame. As a human everybody makes mistake, so try to avoid blaming others. Work as a team to build team spirit.

(16) Finally, Be a Good Observer

Observe things happening around you. Keep track of all major and minor things on your project. Observe the way of developing the code, types of testing and its objective. Observe and understand test progress and make necessary changes if it is off the track in terms of schedule or testing activities. This skill will essential help you to keep yourself updated and ready with course of action for any situation.

Q.1(a) (ii) What is driver and stub? Explain it with an example. [4]

Ans.: Driver and stub [2 marks each]

Stubs are dummy modules that are always distinguish as "called programs", or you can say that is handle in integration testing (top down approach), it used when sub programs are under construction.

Stubs are considered as the dummy modules that always simulate the low level modules.

Drivers are also considered as the form of dummy modules which are always distinguished as "calling programs", that is handled in bottom up integration testing, it is only used when main programs are under construction.
Drivers are considered as the dummy modules that always simulate the high level modules.

Example of Stubs and Drivers is given below: [2 marks]
For Example we have 3 modules login, home, and user module. Login module is ready and need to test it, but we call functions from home and user (which is not ready). To test at a selective module we write a short dummy piece of a code which simulates home and user, which will return values for Login, this piece of dummy code is always called Stubs and it is used in a top down integration.

Considering the same Example above: If we have Home and User modules get ready and Login module is not ready, and we need to test Home and User modules Which return values from Login module, So to extract the values from Login module We write a Short Piece of Dummy code for login which returns value for home and user, So these pieces of code is always called Drivers and it is used in Bottom Up Integration.

So it is fine from the above example that Stubs act "called" functions in top down integration. Drivers are "calling" Functions in bottom up integration.

Q.1(a) (iii) Explain Big-Bang model. List its advantage and disadvantage. [4]

Ans.: Big-Bang model [2 marks]
The Big Bang model is SDLC model where we do not follow any specific process. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer requirement.

Big Bang Model is SDLC model where there is no formal development followed and very little planning is required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.
Usually this model is followed for small projects where the development teams are very small.

Big bang model comprises of focusing all the possible resources in software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to revamp the complete software.

This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It is an ideal model for the product where requirements are not well understood and the final release date is not given.

Big Bang Model advantages and disadvantages [2 marks]

Advantages	Disadvantages
• This is a very simple model. • Little or no planning required. • Easy to manage. • Very few resources required. • Gives flexibility to developers. • Is a good learning aid for new comers or students.	• Very High risk and uncertainty. • Not a good model for complex and object-oriented projects. • Poor model for long and ongoing projects. • Can turn out to be very expensive if requirements are misunderstood.

Q.1(a) (iv) What is test plan? List test planning activities. [4]

Ans.: Test plan [Definition - 1 mark, Test planning activities - 3 marks]
Test plan is the project plan for the testing work to be done. It is not a **test design specification**, a collection of **test cases** or a set of **test procedures**; in fact, most of our test plans do not address that level of detail. Many people have different definitions for test plans.

Testing - like any project, should be driven by a plan. The test plan acts as the anchor for the execution, tracking, and reporting of the entire testing project and covers.

The Test plan is required to plan :

(1) What needs to be tested - the scope of testing, including clear identification of what will be tested and what will not be tested.
(2) How the testing is going to be performed - breaking down the testing into small and manageable tasks and identifying the strategies to be used for carrying out the tasks.
(3) What resources are needed for testing - computer as well as human resources.
(4) The time lines by which the testing activities will be performed.
(5) Risks that may be faced in all of the above, with appropriate mitigation and contingency plans.

Q.1(b) Attempt any ONE of the following : [6]

Q.1(b) (i) Explain GUI testing with suitable example. [6]

Ans.: GUI testing [Explanation – 4 marks, Example – 2 marks]

Graphics User Interface Testing (GUI) Testing is important part of application along with functionality as it affects the usability.

Example: Consider any website like MSBTE, google, yahoo or any login form or GUI of any application to be tested.

It includes following:

- All colors used for background, control colors, and font color have a major impact on users. Wrong color combinations and bright colors may increase fatigue of users.
- All words, Fonts, Alignments, scrolling pages up and down, navigations for different hyperlinks and pages, scrolling reduce usability.
- Error messages and information given to users must be usable to the user. Reports and outputs produced either on screen or printed should be readable. Also paper size on printer, font, size of screen should be consider.
- Screen layout in terms of number of instructions to users, number of controls and number of pages are defined in low level design. More controls on single page and more pages reduce usability.
- Types of control on a single page are very useful considering usability.
- Number of images on page or moving parts on screen may affect performance. These are high-priority defects. It has direct relationships with usability testing, look, and feels of an application. It affects emotions of users and can improve acceptability of an application.

Advantages of GUI Testing:

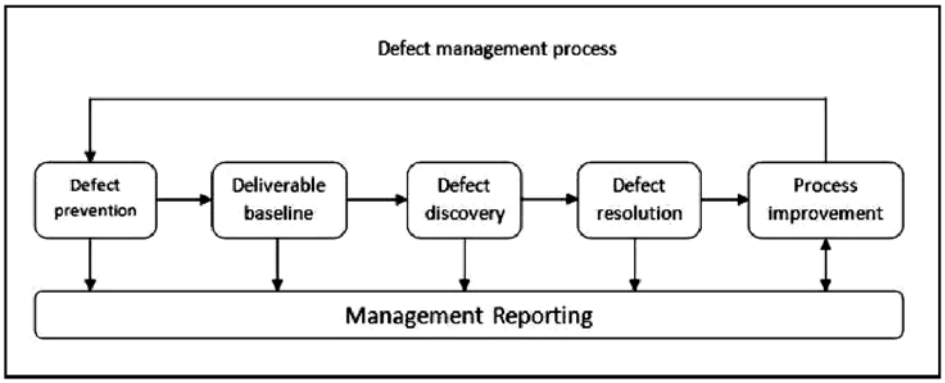
- Good GUI improves feel and look of the application; it psychologically accepts the application by the user.
- GUI represents a presentation layer of an application. Good GUI helps an application due to better experience of the users.
- Consistency of the screen layouts and designs improves usability of an application.

Disadvantages of GUI Testing:

- When number of pages is large and number of controls in a single page is huge.
- Special application testing like those made for blind people or kids below age of five may need special training for testers.

Q.1(b) (ii) Describe steps in Defect Management Process with diagram. [6]

Ans.: [Diagram - 2 marks, Steps - 4 marks]



(1) **Defect Prevention:** Implementation of techniques, methodology and standard processes to reduce the risk of defects.

(2) **Deliverable Baseline:** Deliverables are considered to be ready for further developments. i. e. the deliverables meet exit criteria.

(3) **Defect Discovery:** To find the defect through the process of verification and validation.

(4) **Defect Resolution:** Defect is corrected or corrective action is taken and notification is given to tester.

(5) **Process Improvement:** To identify ways to improve the process to prevent further future occurrences of similar defects, i.e. Corrective and preventive action is taken for processes improvement.

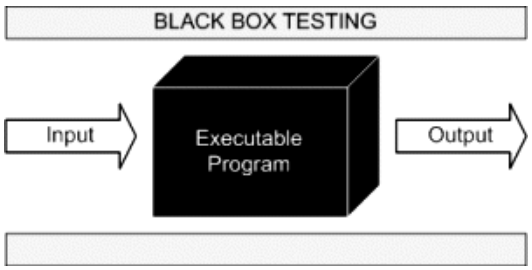
Management Reporting: Reporting is about status of application and processes.

Q.2 Attempt any FOUR of the following : [16]

Q.2(a) Explain Black Box testing? List techniques of Black Box testing. [4]

Ans.: Black Box testing [2 marks]

Black box testing also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see.

A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected.

- It is the testing procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component without reference to its internal structure.
- It is the input/output driven testing techniques because they view the software as a black-box with inputs and outputs, but they have no knowledge of how the system or component is structured inside the box. Here, the tester is concentrating on what the software does, not how, it does it.

This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Behavior or performance errors (non-functional aspects)
- Initialization and termination errors

Techniques of Black Box Testing [2 marks]

- Equivalence Class Partitioning
- Boundary value analysis
- State transition testing
- Use case testing

Q.2(b) How to select a testing tool? Explain in detail. [4]

Ans.: Criteria for Selecting Test Tools [1 mark for each criteria]

The Categories for selecting Test Tools are,

(1) Meeting requirements; (2) Technology expectations;
(3) Training/skills; (4) Management aspects.

(1) Meeting requirements

There are plenty of tools available in the market but rarely do they meet all the requirements of a given product or a given organization. Evaluating different tools for different requirements involve significant effort, money, and time. Given of the plethora of choice available, huge delay is involved in selecting and implementing test tools.

(2) Technology expectations

Test tools in general may not allow test developers to extends/modify the functionality of the framework. So extending the functionality requires going back to the tool vendor and involves additional cost and effort. A good number of test tools require their libraries to be linked with product binaries.

(3) Training/skills

While test tools require plenty of training, very few vendors provide the training to the required level. Organization level training is needed to deploy the test tools, as the user of the test suite are not only the test team but also the development team and other areas like configuration management.

(4) Management aspects

A test tool increases the system requirement and requires the hardware and software to be upgraded. This increases the cost of the already- expensive test tool.

Q.2(c) Prepare four test cases for Admission form for college admission. [4]

Ans.: Consider the college admission form having different fields such as Student's Name, Father's Name, Address, Phone, Caste, admission type, S.S.C percentage, SC Board, Submit button, Reset button. [Any Four - 1 mark each]

Test Case Id	Test case Objectives	Input Data	Expected Result	Actual Result	Status
TC1	Name field	Any name (abcd xyz)	It should accept the name	The name is accepted	Pass
TC2	Phone Field	Any number having less than 10 digits(1234)	It should not accept. Should give error message "Please enter valid phone number"	Error message "Please enter valid phone number"	Pass
TC3	Phone Field	Any Alphabets (abcde)	It should give error message as "Only Numbers"	Error message as "Only Numbers"	Pass

TC4	SSC Percentage Field	65	It should accept	It accepted	Pass
TC5	SSC Percentage Field	30	It should not accept. Should give error message.	Gives error message	Pass
TC6	Address field	Any characters (A-51, Market road, Mumbai)	It should accept.	It accepted	Pass

Q.2(d) What is automation testing also state its pros, cons and applications in detail. [4]

Ans.: Automated Testing [1 mark]

Automated testing is the process through which automated tools run tests that repeat predefined actions, comparing a developing program's expected and actual outcomes. If the program expectations and outcomes align, your project is behaving as it should, and you are likely bug free. If the two don't align, however, there is an issue that needs to be addressed. You'll have to take a look at your code, alter it, and continue to run tests until the actual and expected outcomes align.

Automated testing is good to use when the project is large, there are many system users, or when filling out forms.

Pros of Automated Testing:

(1) Reliable: Tests perform precisely the same operations each time they are run, thereby eliminating human error.

(2) Repeatable: You can test how the software reacts under repeated execution of the same operations.

(3) Programmable: You can program sophisticated tests that bring out hidden information from the application.

(4) Comprehensive: You can build a suite of tests that covers every feature in your application.

(5) Reusable: You can reuse tests on different versions of an application, even if the user interface changes.

(6) Better Quality Software: Because you can run more tests in less time with fewer resources.

(7) Fast: Automated Tools run tests significantly faster than human users.

(8) Economical: As the number of resources for regression test is reduced. Choosing the right tools for the job and targeting the right areas of the organization to deploy them can only realize these benefits. The right areas where the automation fit must be chosen.

Cons of Automated Testing: [1 mark]

(1) Tools can be expensive

The automation tools can be an expensive purchase. As a result, it is important to only use the ones that will give you full, or as close to full coverage, as you can find.

(2) Tools still take time

While the automation process cuts down on the time it takes to test everything by hand, automated testing is still a time intensive process. A considerable amount of time goes into developing the automated tests and letting them run. For example, a large client of ours ran into trouble when their daily run of automated tests exceeded the 24-hour mark.

(3) Tools have limitations

While automated tests will detect most bugs in your system, there are limitations. For example, the automated tools can't test for visual considerations like image color or font size. Changes in these can only be detected by manual testing, which means that not all testing can be done with automatic tools.

Application [1 mark]

Areas where Automation can be attempted first:

(1) Highly redundant tasks or scenarios.
(2) Repetitive tasks that are boring or tend to cause human error.
(3) Well-developed and well-understood use cases or scenarios first.
(4) Relatively stable areas of the application over volatile ones must be automated.

Q.2(e) Draw the diagram of Defect/Bug life cycle and explain its process. [4]

Ans.: Defect/Bug Life cycle: [Diagram – 2 marks, Explanation– 2 marks]

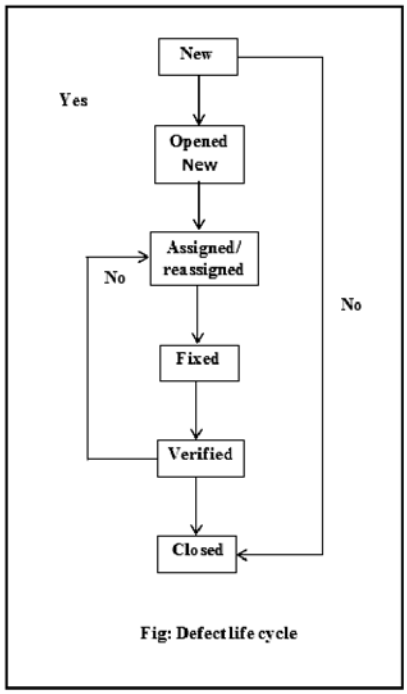
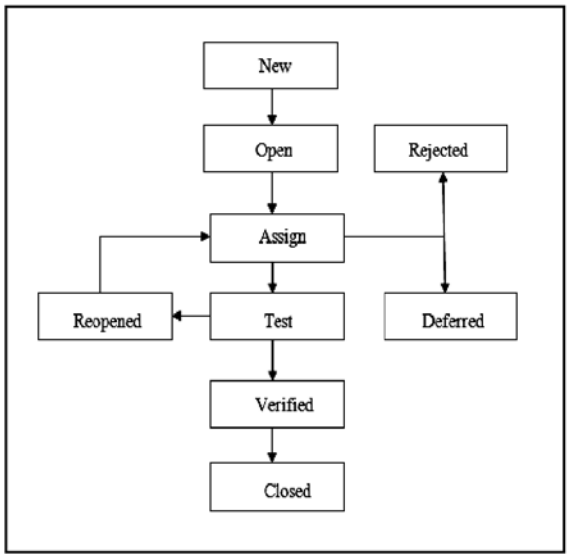


Fig: Defect life cycle

OR



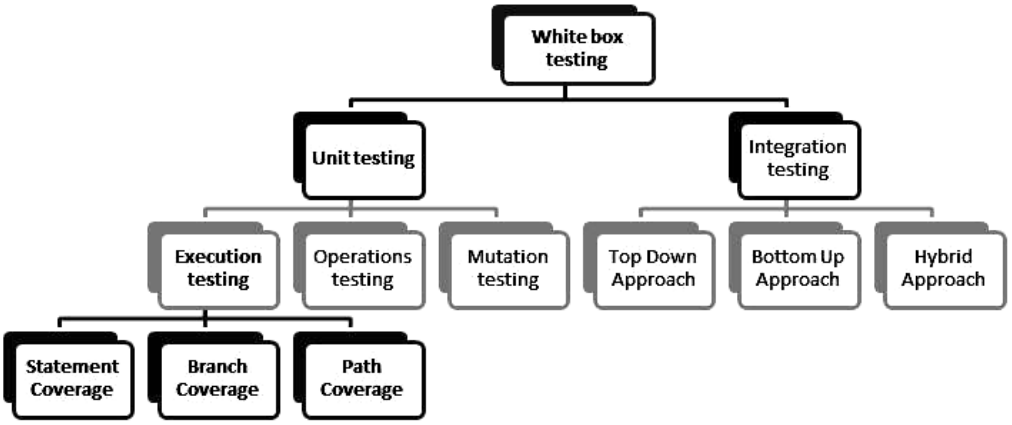
(1) New: When a defect is logged and posted for the first time. It's state is given as new.
(2) Assigned: After the tester has posted the bug, the lead of the tester approves that the bug is genuine and he assigns the bug to corresponding developer and the developer team. It's state given as assigned.
(3) Open: At this state the developer has started analyzing and working on the defect fix.
(4) Fixed: When developer makes necessary code changes and verifies the changes then he/she can make bug status as 'Fixed' and the bug is passed to testing team.
(5) Pending retest: After fixing the defect the developer has given that particular code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest.
(6) Retest: At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.
(7) Verified: The tester tests the bug again after it got fixed by the developer. If the bug is not present in the software, he approves that the bug is fixed and changes the status to "verified".
(8) Reopen: If the bug still exists even after the bug is fixed by the developer, the tester changes the Status to "reopened". The bug goes through the life cycle once again.
(9) Closed: Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to "closed". This tate means that the bug is fixed, tested and approved.
(10) Duplicate: If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to "duplicate".
(11) Rejected: If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "rejected".

(12) Deferred: The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

(13) Not a bug: The state given as "Not a bug" if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of color of some text then it is not a bug but just some change in the looks of the application.

Q.2(f) Draw classification of White Box testing. Explain Structural type of White Box testing in detail. **[4]**

Ans.: Types of White Box Testing [Diagram – 2 marks, Explanation– 2 marks]



$$\text{Statement coverage} = \frac{(\text{Number of statement exercised})}{(\text{Total number of statements}) * 100\%}$$

Different coverage tools may work in slightly different ways, so they may give different coverage figures for the same tests on the same code.

Example

READ A
READ B
IF A > B THEN C = 0
ENDIF

To achieve 100% statement coverage of this code segment just one test case is required, one which ensures that variable A contains a value that is greater than the value of variable B, for ex, A = 12, and B=10. Here we are doing structural test design first, since we are choosing out input values in order to ensure statement coverage.

Example

1. READ A
2. READ B
3. C = A + 2*B
4. IF C > 50 THEN
5. PRINT 'Large C'
6. ENDIF

Each line is regarded as a statement.
TEST SET 1
Test 1_1 : A=2, B=3
Test 1_2 : A=0, B=25
Test 1_3 : A=47, B=1

Which statements we have covered?
In Test 1_1 , the value of C will be 8, so we will cover the statements on lines 1 to 4 and line 6.
In Test 1_2 , the value of C will be 50, so we will cover the statements on lines 1 to 4 and line 6.
In Test 1_3, the value of C will be 49, so we will cover the statements on lines 1 to 4 and line 6.
Since we have covered 5 out of 6 statements, we have 83% statement coverage (with three tests). What test would we need in order to cover statement 5, the one which we have not exercised.
Test 1_4 : A=20, B=25
This time C = 70, so we will print 'Large C' and now we have 100% statement coverage.

Q.3 Attempt any FOUR of the following : [16]

Q.3(a) Explain the code coverage in detail. [4]

Ans.: Code coverage [4 marks]

Code Coverage testing is determining how much code is being tested. It can be calculated using the formula:

$$\text{Code Coverage} = \frac{(\text{Number of lines of code exercised})}{(\text{Total Number of lines of code}) * 100\%}$$

Following are the types of code coverage Analysis:

- Statement coverage and Block coverage
- Function coverage
- Function call coverage
- Branch coverage
- Modified condition/decision coverage

Purpose

Code coverage is a way of ensuring that your tests are actually testing your code. When you run your tests you are presumably checking that you are getting the expected results. Code coverage will tell you how much of your code you exercised by running the test. Your tests may all pass with flying colours, but if you've only tested 50% of your code, how much confidence can you have in it?

There are a number of criteria that can be used to determine how well your tests exercise your code. The most simple is statement coverage, which simply tells you whether you exercised the statements in your code. We will examine statement coverage along with some other coverage criteria a little later.

Using code coverage is a way to try to cover more of the testing problem space so that we come closer to proving the absence of faults, or at least the absence of a certain class of faults.

Q.3(b) Explain static testing in brief. [4]

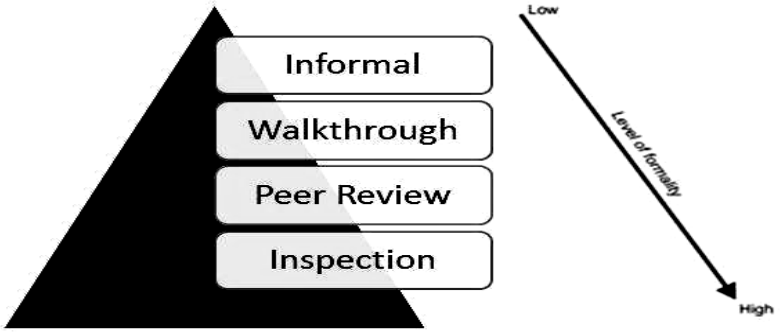
Ans.: Static testing [4 marks]

Static Testing, a software testing technique in which the software is tested without executing the code. It has two parts as listed below:

- Review - Typically used to find and eliminate errors or ambiguities in documents such as requirements, design, test cases, etc.
- Static analysis - The code written by developers are analysed (usually by tools) for structural defects that may lead to defects.

Types of Reviews:

The types of reviews can be given by a simple diagram:

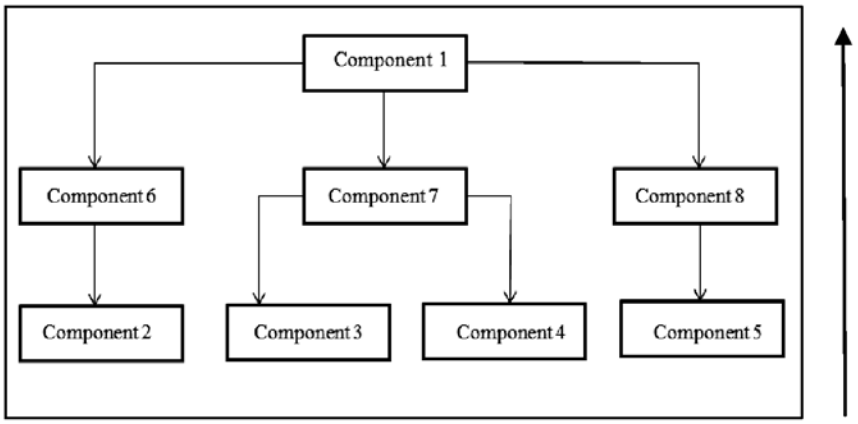


Static testing is a software testing method that involves examination of the program's code and its associated documentation but does not require the program be executed. Dynamic testing, the other main category of software testing methods, involves interaction with the program while it runs. The two methods are frequently used together to try to ensure the functionality of a program.

Static testing may be conducted manually or through the use of various software testing tools. Specific types of static software testing include code analysis, inspection, code reviews and walkthroughs.

Q.3(c) State process of Bi-directional Integration testing with two advantages and two disadvantages. **[4]**

Ans.: Bi-directional integration is a combination of the top-down & bottom-up integration approaches used together to derive integration steps. [Explanation - 2 marks]



As shown in fig, assume that the software components become available in the order mentioned by the component numbers. The individual components 1, 2, 3, 4 and 5 are tested separately and bi-directional integration performed initially with the use of stubs and drivers. Drivers are used to provide upstream connectivity while stubs provide downstream connectivity. A driver is a function which redirects the requests to some other components and stubs simulate the behavior of missing components. After the functionality of these integrated components are tested, the drivers and stubs are discarded. Once components 6, 7 and 8 become available, the integration methodology then focus only on those components, as there are the components which need focus and are new. This approach is also called "Sandwich Integration".

Advantages: [1 mark]

(1) Sandwich approach is useful for very large projects having several subprojects.

(2) Both Top-down and Bottom-up approach starts at a time as per development schedule. Units are tested and brought together to make a system. Integration is done downwards.

Disadvantages: [1 mark]

(1) It require very high cost for testing because one part has Top-down approach while another part has bottom-up approach.

(2) It cannot be used for smaller system with huge interdependence between different modules. It makes sense when the individual subsystem is as good as complete system.

Q.3(d) Describe how to perform security and performance testing with an example. [4]

Ans.: Security Testing [2 marks]

Testers must use a risk-based approach, By identifying risks and potential loss associated with those risks in the system and creating tests driven by those risks, the testers can properly focus on areas of code in which an attack is likely to succeed. Therefore risk analysis at the design level can help to identify potential security problems and their impacts. Once identified ranked, software risks can help guide software security.

Example of a Basic Security Test

This is an example of a very basic security test which anyone can perform on a web site/application:

- Log into the web application.
- Log out of the web application.
- Click the BACK button of the browser (Check if you are asked to log in again or if you are provided the logged-in application.)
- Most types of security testing involve complex steps and out-of-the-box thinking but, sometimes, it is simple tests like the one above that help expose the most severe security risks.

Performance testing [2 marks]

Performance testing is intended to find whether the system meets its performance requirements under normal load or abnormal level of activities. Normal load must be defined by the requirement statement defined by the customer and system design implements them. Performance criteria must be expressed in numerical terms. Design verification can help in determining whether required measures have been taken to meet performance requirements or not. This is one area where verification does not work to that much extents and one needs to test it by actually performing the operation on the system.

Example

For instance, you can test the application network performance on Connection Speed vs. Latency chart. Latency is the time difference between the data to reach from source to destination. Thus, a 70kb page would take not more than 15 seconds to load for a worst connection of 28.8kbps modem (latency=1000 milliseconds), while the page of same size would appear within 5 seconds, for the average connection of 256kbps DSL (latency=100 milliseconds). 1.5mbps T1 connection (latency=50 milliseconds) would have the performance benchmark set within 1 second to achieve this target.

For example, the time difference between the generation of request and acknowledgement of response should be in the range of x ms (milliseconds) and y ms, where x and y are standard digits. A successful performance testing should project most of the performance issues, which could be related to database, network, software, hardware etc…

Q.3(e) Define metrics and measurements. Explain need of software measurement. [4]

Ans.: Metrics and measurement [2 marks]

Metrics is a relative measurement of status of process or product in terms of two or more entities taken together for comparison. Measurements are key element for controlling software engineering processes.

Need of software measurements [2 marks]

(1) Understanding: Metrics can help in making the aspects of process more visible, thereby giving a better understanding of the relationship among the activities and entities they affect.

(2) Control: Using baselines, goals and an understanding of the relationships, we can predict what is likely to happen and correspondingly, make appropriate changes in the process to help meet the goals.

(3) Improvement: By taking corrective actions and making appropriate changes, we can improve a product. Similarly, based on the analysis of a project, a process can also be improved.

Q.4(a) Attempt any THREE of the following : [12]

Q.4(a) (i) Differentiate between Alpha Testing and Beta Testing in brief. [4]

Ans.: [Any Four - 1 mark each]

	Alpha Testing	**Beta Testing (Field Testing)**
(1)	It is always performed by the developers at the software development site.	It is always performed by the customers at their own site.
(2)	Sometimes it is also performed by Independent Testing Team.	It is not performed by Independent Testing Team.
(3)	Alpha Testing is not open to the market and public	Beta Testing is always open to the market and public.
(4)	It is conducted for the software application and project.	It is usually conducted for software product.
(5)	It is always performed in Virtual Environment.	It is performed in Real Time Environment.
(6)	It is always performed within the organization.	It is always performed outside the organization.
(7)	It is the form of Acceptance Testing.	It is also the form of Acceptance Testing.
(8)	Alpha Testing is definitely performed and carried out at the developing organizations location with the involvement of developers.	Beta Testing (field testing) is performed and carried out by users or you can say people at their own locations and site using customer data.
(9)	It comes under the category of both White Box Testing and Black Box Testing.	It is only a kind of Black Box Testing.
(10)	Alpha Testing is always performed at the time of Acceptance Testing when developers test the product and project to check whether it meets the user requirements or not.	Beta Testing is always performed at the time when software product and project are marketed.
(11)	It is always performed at the developer's premises in the absence of the users.	It is always performed at the user's premises in the absence of the development team.
(12)	Alpha Testing is not known by any other different name.	Beta Testing is also known by the name Field Testing means it is also known as field testing.
(13)	It is considered as the User Acceptance Testing (UAT) which is done at developer's area.	It is also considered as the User Acceptance Testing (UAT) which is done at customers or users area.

Q.4(a) (ii) Explain static and dynamic testing tools in details. [4]

Ans.: Static and dynamic testing tools [2 marks each]

Static testing tools are used during static analysis of a system. Static testing tools are used throughout a software development life cycle, e.g , tools used for verification purposes. There are many varieties of static testing tools used by different people as per the type of system being developed.

Code complexity measurement tools can be used to measure the complexity of a given code. Similarly, data-profiling tools can be used to optimize a database. Code-profiling tools can be used to optimize code. Test-generators are used for generating a test plan form code. Syntax-checking tools are used to verify correctness of code.

Dynamic testing tools are used at different levels of testing starting from unit testing & which may go up to system testing & performance testing. These tools are generally used by tester. There are many different tools used for dynamic testing. Some of the areas covered by testing tools are:

(1) Regression testing using automated tools.
(2) Defect tracking and communication systems used by tracking & communication. Performance, Load, stress-testing tools.

Q.4(a) (iii) Why is it essential to setup criteria for testing? List any three criteria in different situations. [4]

Ans.: There is a need to setup criteria for testing because: [Any Two - 1 mark]

(i) An early start to testing reduces the cost, time to rework and error free software that is delivered to the client.
(ii) Software Development Life Cycle (SDLC) testing can be started from the Requirements Gathering phase and lasts till the deployment of the software.
(iii) It also depends on the development model that is being used. For example in Water fall model formal testing is conducted in the Testing phase, but in incremental model, testing is performed at the end of every increment/iteration and at the end the whole application is tested.
(iv) Testing is done in different forms at every phase of SDLC like during Requirement gathering phase, the analysis and verification of requirements are also considered testing.
(v) Reviewing the design in the design phase with intent to improve the design is also considered as testing.
(vi) Testing performed by a developer on completion of the code is also categorized as Unit type of testing.

Any 3 criteria's in different situation are : [1 mark each]

(1) Start with the static white box testing procedure when the specifications of the software to be developed are ready.
(2) Use the code coverage analyzer to test whether the whole code is getting executed and covered.
(3) Perform unit testing as soon as one of the unit or sub module in the software is ready.

Q.4(a) (iv) Explain test deliverables in details. [4]

Ans.: Test deliverables [4 marks]

The test plan also identifies the deliverables that should come out of the test cycle/testing activity. The Deliverables includes the following, all reviewed and approved by the appropriate people.

(i) The test plan itself master test plan and previous other test plans for the project)
(ii) Test case design specifications.
(iii) Test cases, including any automation that is specified in the plan.

(iv) Test logs produced by running the tests.
(v) Test summary reports

Q.4(b) Attempt any ONE of the following : [6]

Q.4(b) (i) Describe V-model with labelled diagram. State it's any two advantages and disadvantages. Also write where it is applicable. [6]

Ans.: V-model with labelled diagram [Explanation with diagram - 3 marks]

V model means verification and validation model. It is sequential path of execution of processes. Each phase must be completed before the next phase begins.

Under V-model, the corresponding testing phase of the development phase is planned in parallel. So there is verification on one side of V & validation phase on the other side of V.

Verification Phase:

1. **Overall Business Requirement:** In this first phase of the development cycle, the product requirements are understood from customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirements. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.
2. **Software Requirement:** Once the product requirements are clearly known, the system can be designed. The system design comprises of understanding & detailing the complete hardware, software & communication set up for the product under development. System test plan is designed based on system design. Doing this at earlier stage leaves more time for actual test execution later.
3. **High level design:** High level specification are understood & designed in this phase. Usually more than one technical approach is proposed & based on the technical & financial feasibility, the final decision is taken. System design is broken down further into modules taking up different functionality.
4. **Low level design:** In this phase the detailed integral design for all the system modules is specified. It is important that the design is compatible with the other modules in the system & other external system. Components tests can be designed at this stage based on the internal module design,
5. **Coding:** The actual coding of the system modules designed in the design phase is taken up in the coding phase. The base suitable programming language is decided base on requirements. Coding is done based on the coding guidelines & standards.

Validation:

1. Unit Testing: Unit testing designed in coding are executed on the code during this validation phase. This helps to eliminate bugs at an early stage.
2. Components testing: This is associated with module design helps to eliminate defects in individual modules.
3. Integration Testing: It is associated with high level design phase & it is performed to test the coexistence & communication of the internal modules within the system
4. System Testing: It is associated with system design phase. It checks the entire system functionality & the communication of the system under development with external systems. Most of the software & hardware compatibility issues can be uncovered using system test execution.

Acceptance Testing: It is associated with overall & involves testing the product in user environment. These tests uncover the compatibility issues with the other systems available in the user environment. It also uncovers the non-functional issues such as load & performance defects in the actual user environment.

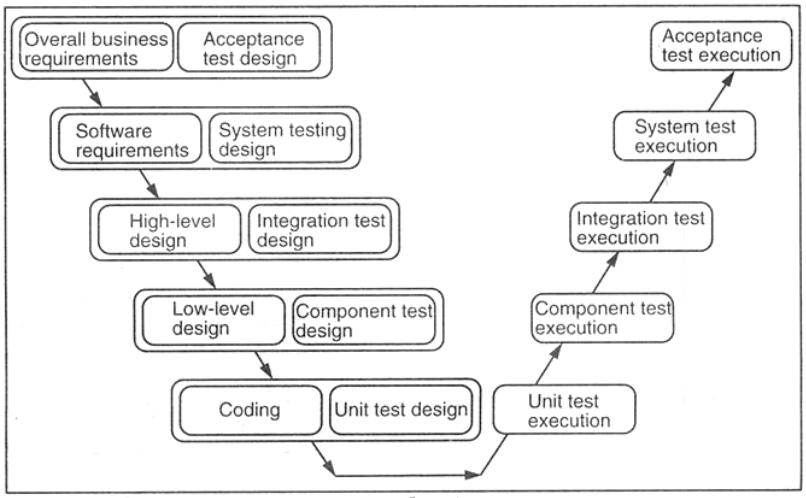


Fig.: V- Model

Advantages of V-model: [Two advantages - 1 mark]

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model: [Two disadvantages - 1 mark]

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

Application: [Explanation of where it is applicable - 1 mark]

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

Q.4(b) (ii) With the help of example explain Boundary Value Analysis. [6]

Ans.: Boundary Value Analysis [Explanation - 4 marks, Example - 2 marks]

Most of the defects in software products hover around conditions and boundaries. By conditions, we mean situations wherein, based on the values of various variables, certain actions would have to be taken. By boundaries, we mean "limits" of values of the various variables.

- This is one of the software testing technique in which the test cases are designed to include values at the boundary. If the input data is used within the boundary value limits, then it is said to be Positive Testing. If the input data is picked outside the boundary value limits, then it is said to be Negative Testing.
- Boundary value analysis is another black box test design technique and it is used to find the errors at boundaries of input domain rather than finding those errors in the center of input.
- Each boundary has a valid boundary value and an invalid boundary value. Test cases are designed based on the both valid and invalid boundary values. Typically, we choose one test case from each boundary.
- Same examples of Boundary value analysis concept are:

One test case for exact boundary values of input domains each means 1 and 100. One test case for just below boundary value of input domains each means 0 and 99. One test case for just above boundary values of input domains each means 2 and 101.

- **For Example:** A system can accept the numbers from 1 to 10 numeric values. All other numbers are invalid values. Under this technique, boundary values 0, 1, 2, 9, 10, 11 can be tested.
- Another Example is in exam has a pass boundary at 40 percent, merit at 75 percent and distinction at 85 percent. The Valid Boundary values for this scenario will be as follows:
  49, 50 - for pass
  74, 75 - for merit
  84, 85 - for distinction
  Boundary values are validated against both the valid boundaries and invalid boundaries. The Invalid Boundary Cases for the above example can be given as follows
  0 - for lower limit boundary value
  101 - for upper limit boundary value
- Boundary value analysis is a black box testing and is also applies to white box testing. Internal data structures like arrays, stacks and queues need to be checked for boundary or limit conditions; when there are linked lists used as internal structures, the behavior of the list at the beginning and end have to be tested thoroughly.
- Boundary value analysis help identify the test cases that are most likely to uncover defects.

Q.5 Attempt any TWO of the following : [16]

Q.5(a) Which parameters are considered while writing good defect report? Also write contents of defect template. [8]

Ans.: [Parameters - 4 marks, Contents of defect template - 4 marks]

A defect report documents an anomaly discovered during testing. It includes all the information needed to reproduce the problem, including the author, release/build number, open/close dates, problem area, problem description, test environment, defect type, how it was detected, who detected it, priority, severity, status, etc.

After uncovering a defect (bug), testers generate a formal defect report. The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.

Defect Report Template

In most companies, a defect reporting tool is used and the elements of a report can vary. However, in general, a defect report can consist of the following elements.

ID	Unique identifier given to the defect. (Usually Automated)
Project	Project name
Product	Product name
Release Version	Release version of the product. (e.g. 1.2.3)
Module	Specific module of the product where the defect was detected.
Detected Build Version	Build version of the product where the defect was detected (e.g. 1.2.3.5)
Summary	Summary of the defect. Keep this clear and concise.
Description	Detailed description of the defect. Describe as much as possible but without repeating anything or using complex words. Keep it simple but comprehensive.
Steps to Replicate	Step by step description of the way to reproduce the defect. Number the steps.
Actual Result	The actual result you received when you followed the steps.
Expected Results	The expected results.

Attachments	Attach any additional information like screenshots and logs.
Remarks	Any additional comments on the defect.
Defect Severity	Severity of the Defect.
Defect Priority	Priority of the Defect.
Reported By	The name of the person who reported the defect.
Assigned To	The name of the person that is assigned to analyze/fix the defect.
Status	The status of the defect.
Fixed Build Version	Build version of the product where the defect was fixed (e.g. 1.2.3.9)

Q.5(b) With the help of example explain Equivalence partitioning. [8]

Ans.: Equivalence partitioning [Explanation & Example - 4 marks each]

- Equivalence Partitioning also called as equivalence class partitioning. It is abbreviated as ECP. It is a software testing technique that divides the input test data of the application under test into each partition.
- An advantage of this approach is it reduces the time required for performing testing of a software due to less number of test cases.
- In short it is the process of taking all possible test cases and placing them into classes. One test value is picked from each class while testing.
- **Example**
  If you are testing for an input box accepting numbers from 1 to 1000 then there is no use in writing thousand test cases for all 1000 valid input numbers plus other test cases for invalid data.
- Using equivalence partitioning method above test cases can be divided into three sets of input data called as classes. Each test case is a representative of respective class.
- So in above example we can divide our test cases into three equivalence classes of some valid and invalid inputs.
- **Test cases for input box accepting numbers between 1 and 1000 using Equivalence Partitioning:**
  (1) One input data class with all valid inputs. Pick a single value from range 1 to 1000 as a valid test case. If you select other values between 1 and 1000 then result is going to be same. So one test case for valid input data should be sufficient.
  (2) Input data class with all values below lower limit. I.e. any value below 1, as a invalid input data test case.
  (3) Input data with any value greater than 1000 to represent third invalid input class.
- So using equivalence partitioning you have categorized all possible test cases into three classes. Test cases with other values from any class should give you the same result.
- We have selected one representative from every input class to design our test cases. Test case values are selected in such a way that largest number of attributes of equivalence class can be exercised.
- Equivalence partitioning uses fewest test cases to cover maximum requirements.

Q.5(c) How test case specifications useful in designing test cases? [8]

Ans.: [Specifications (any four) – 4 marks, their justification - 4 marks]

The test case specifications should be developed from the test plan and are the second phase of the test development life cycle. The test specification should explain "how" to implement the test cases described in the test plan. Test case specifications are useful as it enlists the specification details of the items.

Test Specification Items are must for each test specification should contain the following items:

1. Case No.: The test case number should be a three digit identifier of the following form: c.s.t, where: c- is the chapter number, s- is the section number, and t- is the test case number.
2. Title: is the title of the test.

3. Programme: is the program name containing the test.
4. Author: is the person who wrote the test specification.
5. Date: is the date of the last revision to the test case.
6. Background: (Objectives, Assumptions, References, Success Criteria): Describes in words how to conduct the test.
7. Expected Error(s): Describes any errors expected
8. Reference(s): Lists reference documentation used to design the specification.
9. Data: (Tx Data, Predicted Rx Data): Describes the data flows between the Implementation under Test (IUT) and the test engine.
10. Script: (Pseudo Code for Coding Tests): Pseudo code (or real code) used to conduct the test.

Q.6 Attempt any FOUR of the following : [16]

Q.6(a) What is defect management? Give defect classification in detail. [4]

Ans.: Defect management [1 mark]

It is the process of it enhances quality by adding value to the most important attributes of software like reliability, maintainability, efficiency and portability.

Defect Classification: [Any three - 1 mark each]

Requirements and specification defect: Requirement related defects arise in a product when one fails to understand what is required by the customer. These defects may be due to customer gap, where the customer is unable to define his requirements, or producer gap, where developing team is not able to make a product as per requirements. Defects injected in early phases can persist and be very difficult to remove in later phases. Since any requirements documents are written using natural language representation, there are very often occurrences of ambiguous, contradictory, unclear, redundant and imprecise requirements. Specifications are also developed using natural language representations.

Design Defects: Design defects occur when system components, interactions between system components, interactions between the outside software/hardware, or users are incorrectly designed. This covers in the design of algorithms, control, logic/ data elements, module interface descriptions and external software/hardware/user interface descriptions. Design defects generally refer to the way of design creation or its usage while creating a product. The customer may or may not be in a position to understand these defects, if structures are not correct. They may be due to problems with design creation and implementation during software development life cycle.

Coding Defects: Coding defects may arise when designs are implemented wrongly. If there is absence of development/coding standards or if they are wrong, it may lead to coding defects. Coding defects are derived from errors in implementing the code. Coding defect classes are closely related to design defect classes especially if pseudo code has been used for detailed design. Some coding defects come from a failure to understand programming language constructs, and miscommunication with the designers. Others may have transcription or omission origins. At times it may be difficulty to classify a defect as a design or as a coding detect.

Testing Defect: Testing defect are defects introduced in an application due to wrong testing, or defects in the test artifact leading to wrong testing. Defects which cannot be reproduced, or are not supported by requirement or are duplicate may represent a false call .In this defects includes

(1) Test-design defect: test-design defect refers to defects in test artifacts, there can be defects in test plans, test scenarios, test cases and test data definition which can lead to defect in software.

(2) Test-environment defect: this defect may arise when test environment is not set properly, test environment may be comprised of hardware, software, simulator and people doing testing.

(3) Test-tool defects: any defects introduced by a test tool may be very difficult to find and resolve, as one may have to find the defect using manual test as against automated tools.

Q.6(b) Give difference between Quality Assurance and Quality Control. [4]

Ans.: [Any four differences - 4 marks]

Quality Assurance	Quality Control
Process oriented activities.	Product oriented activities.
QA is the process of managing for quality.	QC is used to verify the quality of the output.
They measure the process, identify the deficiencies/weakness and suggest improvements.	They measure the product, identify the deficiencies/weakness and suggest improvements.
Relates to all products that will ever be created by a process.	Relates to specific product.
Activities of QA are Process Definition and Implementation, Audits and Training	Activities of QC are Reviews and Testing.
Verification is an example of QA	Validation/Software Testing is an example of QC
Preventive activities.	It is a corrective process.
Quality assurance is a proactive process.	Quality control is a reactive process.
QA is a managerial tool	QC is a corrective tool

Q.6(c) State any four objectives of user documentation testing. How these are useful in planning user documentation test? [4]

Ans.: Documentation testing is a non-functional type of software testing.

[Objectives - 2 marks, Useful in planning - 2 marks]

(i) It is a type of non-functional testing.

(ii) Any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures, or results'. Documentation is as important to a product's success as the product itself. If the documentation is poor, non-existent, or wrong, it reflects on the quality of the product and the vendor.

(iii) As per the IEEE Documentation describing plans for, or results of, the testing of a system or component, Types include test case specification, test incident report, test log, test plan, test procedure, test report. Hence the testing of all the above mentioned documents is known as documentation testing.

(iv) This is one of the most cost effective approaches to testing. If the documentation is not right: there will be major and costly problems. The documentation can be tested in a number of different ways to many different degrees of complexity. These range from running the documents through a spelling and grammar checking device, to manually reviewing the documentation to remove any ambiguity or inconsistency.

(v) Documentation testing can start at the very beginning of the software process and hence save large amounts of money, since the earlier a defect is found the less it will cost to be fixed.

Q.6(d) Write 4 test cases for user login form. [4]

Ans.: [Any four test case - 1 mark each]

TC_Id	TC name	Steps	Input data	Expected result	Actual Result	Status
TC_01	User name	Enter the username in alphanumeric alphabets A-Z Number 0-9	"abc123"	It should accept the username	It is accepted username	Pass
TC_02	Password	Enter the password in alphanumeric alphabets A-Z Number 0-9	"abc123"	It should accept the password	It is accepted password	Pass

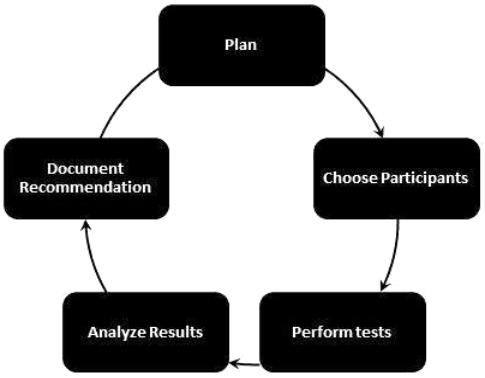
TC_03	Submit	1. After valid username and password 2. Click on submit button		It should goes to next page	It is going to next page	Pass
TC_04	Cancel	Click on cancel button		It should remain in login page with blank fields	It shows login page with blank fields	Pass

Q.6(e) Describe following testing with example. **[4]**

(i) Usability testing **(ii) Compatibility testing**

Ans.: (i) Usability Testing [2 marks]

Usability testing, a non-functional testing technique that is a measure of how easily the system can be used by end users. It is difficult to evaluate and measure but can be evaluated based on the below parameters:



- Level of Skill required to learn/use the software. It should maintain the balance for both novice and expert user.
- Time required to get used to in using the software.
- The measure of increase in user productivity if any.
- Assessment of a user's attitude towards using the software.

(ii) Compatibility Testing [2 marks]

Compatibility testing is a non-functional testing conducted on the application to evaluate the application's compatibility within different environments. It can be of two types - forward compatibility testing and backward compatibility testing.

- Operating system Compatibility Testing - Linux , Mac OS, Windows
- Database Compatibility Testing - Oracle SQL Server
- Browser Compatibility Testing - IE , Chrome, Firefox
- Other System Software - Web server, networking/ messaging tool, etc.

❑ ❑ ❑ ❑ ❑