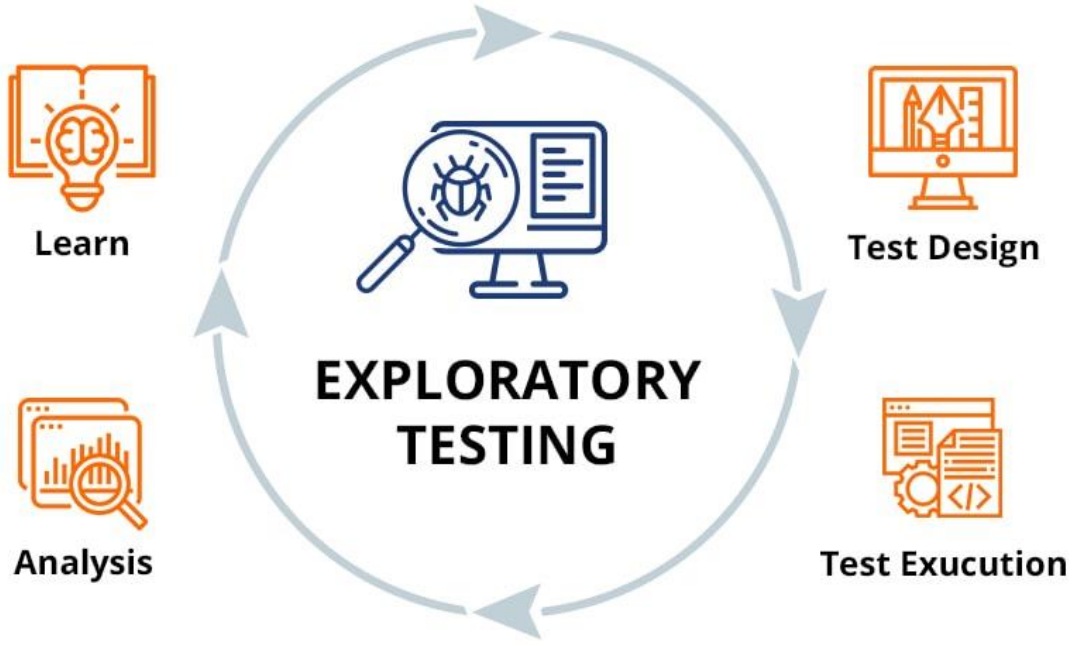
TOPS TECHNOLOGIES

Training | Outsourcing | Placement | Study Abroad

Software Testing Assignment - 2

1. What is Exploratory Testing?

- Exploratory testing allows you to think outside the box and come up with use cases that might not be covered in a test case. For example, you might perform one test and then ask yourself, “What if I tried this? What if I didn’t do that?”
- Some defects, typically the hard ones to find, are dependent on a sequence of events. If you don’t have really deep test cases, you can miss finding defects that exploratory testing can find in a less structured, but longer, test session.



Exploratory test management needs to ensure that the following happens:

- A clear mission of the test is established.
- Thorough notes are taken of what needs to be tested, why, and the assessed quality of the application.
- Issues and questions raised during testing are tracked.
- Two testers are appropriately assigned based on their experience.
- This is accomplished through 5 phases, or session based test management (SBTM Cycle):

1. Classify the bugs.

- Categorize the kinds of problems most commonly found in past projects.
- Seek the root cause of these problems.
- Identify the risks and develop ideas to test the application.

2. Create a test charter.

- Identify what to test, how it can be tested, and what factors to consider.
- Describe the starting point of testing.
- Define how users will utilize the system.

3. Create a time box.

- Two testers work together for at least 90 minutes.
- Interruptions do not occur during the 90 minutes.
- An extension or reduction of 45 minutes is acceptable.
- Testers react to the response of the application and prepare for the correct outcome.

4. Review the results.

- Assess the defects.
- Learn from the test.
- Analyze coverage areas.

5. Debrief.

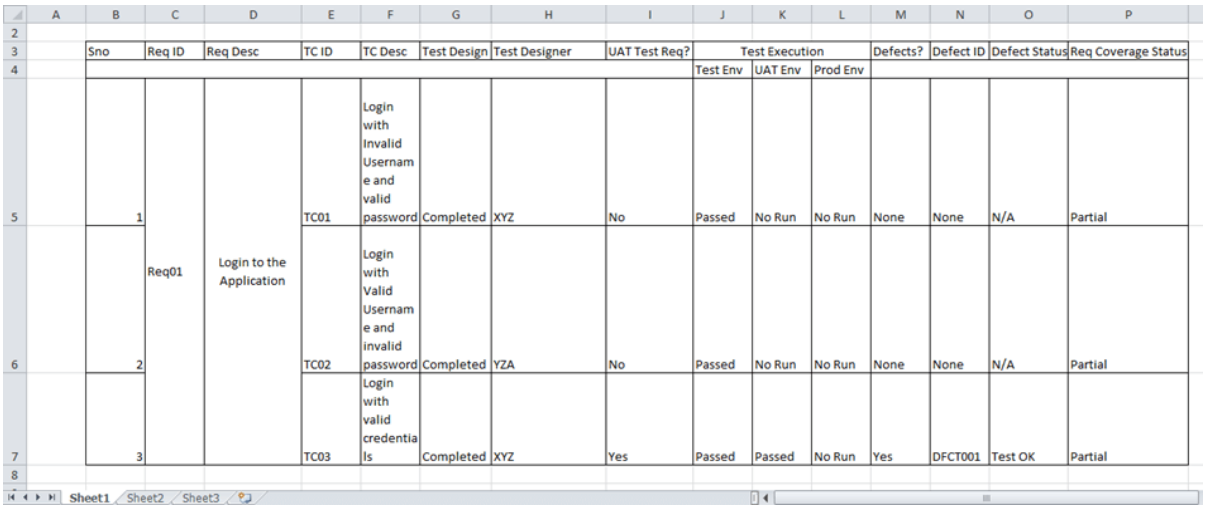
- Compile results.
- Compare results to the character.
- Check for needed additional testing.

2. What is Traceability Matrix?

- A Traceability Matrix is a document that co-relates any two-baseline documents that require a many-to-many relationship to check the completeness of the relationship.
- It is used to track the requirements and to check the current project requirements are met.
- **Which Parameters to include in Requirement Traceability Matrix?**
- Requirement ID
- Requirement Type and Description
- Test Cases with Status

Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

- Above is a sample requirement traceability matrix.
- But in a typical software testing project, the traceability matrix would have more than these parameters.



Sno	Req ID	Req Desc	TC ID	TC Desc	Test Design	Test Designer	UAT Test Req?	Test Execution			Defects?	Defect ID	Defect Status	Req Coverage Status
								Test Env	UAT Env	Prod Env				
1	Req01	Login to the Application	TC01	Login with Invalid Username and valid password	Completed	XYZ	No	Passed	No Run	No Run	None	None	N/A	Partial
2			TC02	Login with Valid Username and invalid password	Completed	YZA	No	Passed	No Run	No Run	None	None	N/A	Partial
3			TC03	Login with valid credentials	Completed	XYZ	Yes	Passed	Passed	No Run	Yes	DFCT001	Test OK	Partial

- As illustrated above, a requirement traceability matrix can:
- Show the requirement coverage in the number of test cases
- Design status as well as execution status for the specific test case
- If there is any User Acceptance test to be done by the users, then UAT status can also be captured in the same matrix.
- The related defects and the current state can also be mentioned in the same matrix.
- This kind of matrix would be providing **One Stop Shop** for all the testing activities.
- Apart from maintaining an excel separately. A testing team can also opt for requirements tracing available Test Management Tools.

- **Types of Traceability Test Matrix**
- In Software Engineering, traceability matrix can be divided into three major components as mentioned below:
- **Forward traceability**: This matrix is used to check whether the project progresses in the desired direction and for the right product. It makes sure that each requirement is applied to the product and that each requirement is tested thoroughly. It maps requirements to test cases.
- **Backward or reverse traceability:** It is used to ensure whether the current product remains on the right track. The purpose behind this type of

traceability is to verify that we are not expanding the scope of the project by adding code, design elements, test or other work that is not specified in the requirements. It maps test cases to requirements.

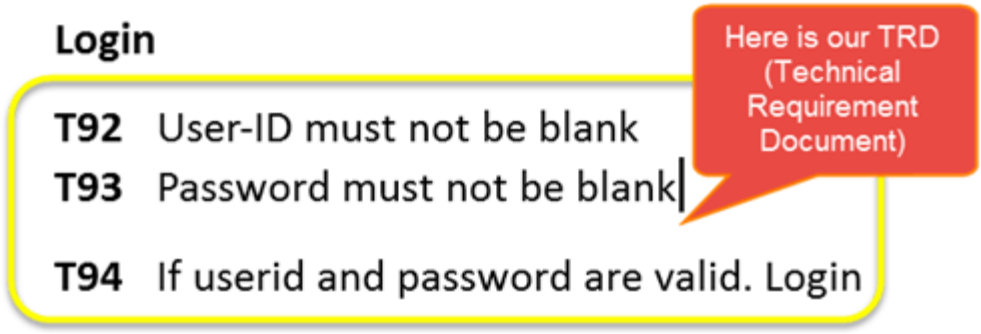
- **Bi-directional traceability (Forward+Backward):** This traceability matrix ensures that all requirements are covered by test cases. It analyzes the impact of a change in requirements affected by the Defect in a work product and vice versa.

- **How to create Requirement Traceability Matrix**
- Let's understand the concept of Requirement Traceability Matrix through a Guru99 banking project.
- On the basis of **the Business Requirement Document (BRD)** and **Technical Requirement Document (TRD)**, testers start writing test cases.
- Let suppose, the following table is our Business Requirement Document or BRD for **Guru99 banking project**.
- Here the scenario is that the customer should be able to login to Guru99 banking website with the correct password and user#id while manager should be able to login to the website through customer login page.

BR#	Module Name	Applicable Roles	Description
B1	Login and Logout	Manager Customer	**Customer:** A customer can login using the login page **Manager:** A manager can login using the login page f customer. Post Login homepage will show ifferent links based on role
B2	Enquiry	Customer	**ustomer:** A customer can have multiple bank accounts. He can view balance of his accounts only **Manager:** A manager can view balance of all the customers who come under his supervision
B3	Fund Transfer	Manager Customer	**Customer:** A customer can have transfer funds from his "own" account to any destination account. **Manager:** A manager can transfer funds from any

Business Requirement # for Guru99 banking project

- While the below table is our **Technical Requirement Document (TRD)**.

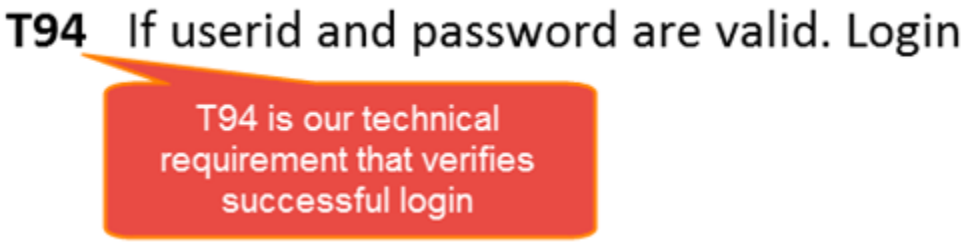


- **Note:** QA teams do not document the BRD and TRD. Also, some companies use **Function Requirement Documents (FRD)** which are similar to Technical Requirement Document but the process of creating Traceability Matrix remains the same.
- Let's Go Ahead and create RTM in Testing
- **Step 1:** Our sample Test Case is
- "Verify Login, when correct ID and Password is entered, it should log in successfully"

TestCase #	Test Case	Test Steps	Test Data	Expected Result
1	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= Guru99 pass= 1234	Login Successful

When correct password and id entered, it should login successfully

- **Step 2**: Identify the Technical Requirement that this test case is verifying. For our test case, the technical requirement is T94 is being verified.



- **Step 3:** Note this Technical Requirement (T94) in the Test Case.

-

TestCase #	TR #		Test Steps	Test Data	Expected
1	T94	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= Guru99 pass= 1234	Login Successful

Note the Technical Requirement in the test case

- **Step 4:** Identify the Business Requirement for which this TR (Technical Requirement-T94) is defined
-

BR#	Module Name	Applicable Roles	Description
B1	Login and Logout	Manager Customer	**Customer:** A customer can login using the login page **Manager:** A manager can login using the login page of customer. Post Login homepage will show different links based on role

Identify the Business Requirement for which T94 is defined

- **Step 5:** Note the BR (Business Requirement) in Test Case
-

TestCase #	BR #	TR #	Test Case	Test Steps	Test Data	Expe
1	B1	T94	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= Guru99 pass= 1234	Login Successful

- **Step 6:** Do above for all Test Cases. Later Extract the First 3 Columns from your Test Suite. RTM in testing is Ready!

Business Requirement #	Technical Requirement #	Test Case ID
B1	T94	1
B2	T95	3
B3	T96	3
B4	T97	4

Requirement Traceability Matrix

- **Advantage of Requirement Traceability Matrix**
- It confirms 100% test coverage
- It highlights any requirements missing or document inconsistencies
- It shows the overall defects or execution status with a focus on business requirements

- It helps in analyzing or estimating the impact on the QA team's work with respect to revisiting or re-working on the test cases.

3. What is Boundary value testing?

- Boundary Value Analysis is based on testing the boundary values of valid and invalid partitions. The behavior at the edge of the equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.
- It checks for the input values near the boundary that have a higher chance of error. Every partition has its maximum and minimum values, and these maximum and minimum values are the boundary values of a partition.
- A boundary value for a valid partition is a valid boundary value.
- A boundary value for an invalid partition is an invalid boundary value.
- For each variable we check-
    - Minimum value.
    - Just above the minimum.
    - Nominal Value.
    - Just below Max value.
    - Max value.
- **Example:** Consider a system that accepts ages from 18 to 56.

Boundary Value Analysis(Age accepts 18 to 56)		
Invalid (min-1)	Valid (min, min + 1, nominal, max – 1, max)	Invalid (max + 1)
17	18, 19, 37, 55, 56	57

- **Valid Test cases:** Valid test cases for the above can be any value entered greater than 17 and less than 57.

- Enter the value- 18.
- Enter the value- 19.
- Enter the value- 37.
- Enter the value- 55.
- Enter the value- 56.
- **Invalid Testcases:** When any value less than 18 and greater than 56 is entered.
- Enter the value- 17.
- Enter the value- 57.
- **Single Fault Assumption:** When more than one variable for the same application is checked then one can use a single fault assumption. Holding all but one variable to the extreme value and allowing the remaining variable to take the extreme value. For n variable to be checked:
- ***Maximum of 4n+1 test cases***
- **Problem:** Consider a Program for determining the Previous Data.
- ***Input:*** *Day, Month, Year with valid ranges as-*
- *1 ≤ Month≤12*
  1 ≤ Day ≤31
  1900 ≤ Year ≤ 2000
- Design Boundary Value Test Cases.
- **Solution:** Taking the year as a Single Fault Assumption i.e. year will be having values varying from 1900 to 2000 and others will have nominal values.

Test Cases	Month	Day	Year	Output
1	6	15	1990	14 June 1990
2	6	15	1901	14 June 1901

3	6	15	1960	14 June 1960
4	6	15	1999	14 June 1999
5	6	15	2000	14 June 2000

Taking Day as Single Fault Assumption i.e. Day will be having values varying from 1 to 31 and others will have nominal values.

Test Case	Month	Day	Year	Output
6	6	1	1960	31 May 1960
7	6	2	1960	1 June 1960
8	6	30	1960	29 June 1960
9	6	31	1960	Invalid day

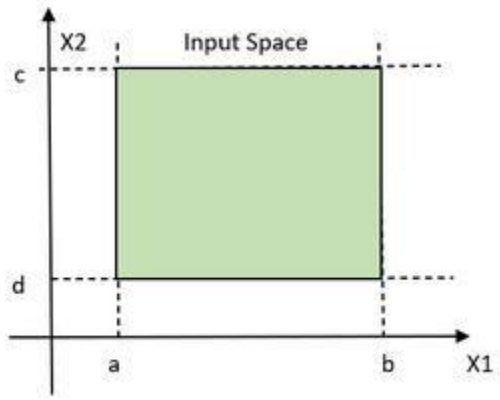
Taking Month as Single Fault Assumption i.e. Month will be having values varying from 1 to 12 and others will have nominal values.

Taking Month as Single Fault Assumption i.e. Month will be having values varying from 1 to 12 and others will have nominal values.

Test Case	Month	Day	Year	Output
10	1	15	1960	14 Jan 1960
11	2	15	1960	14 Feb 1960
12	11	15	1960	14 Nov 1960
13	12	15	1960	14 Dec 1960

For the n variable to be checked Maximum of 4n + 1 test case will be required. Therefore, for n = 3, the maximum test cases are-

The focus of BVA: BVA focuses on the input variable of the function. Let's define two variables X1 and X2, where X1 lies between a and b and X2 lies between c and d.



Showing legitimate domain

The idea and motivation behind BVA are that errors tend to occur near the extremes of the variables. The defect on the boundary value can be the result of countless possibilities.

Typing of Languages: BVA is not suitable for free-form languages such as COBOL and FORTRAN, These languages are known as weakly typed languages. This can be useful and can cause bugs also.

PASCAL, ADA is the strongly typed language that requires all constants or variables defined with an associated data type.

Limitation of Boundary Value Analysis:

- It works well when the product is under test.
- It cannot consider the nature of the functional dependencies of variables.
- BVA is quite rudimentary.

4. What is Equivalence Partitioning testing?

- It is a type of black-box testing that can be applied to all levels of software testing. In this technique, input data are divided into the equivalent partitions that can be used to derive test cases-
- In this input data are divided into different equivalence data classes.
- It is applied when there is a range of input values.
- **Example:** Below is the example to combine Equivalence Partitioning and Boundary Value.
- Consider a field that accepts a minimum of 6 characters and a maximum of 10 characters. Then the partition of the test cases ranges 0 – 5, 6 – 10, 11 – 14.

Test Scenario	Test Description	Expected Outcome
1	Enter value 0 to 5 character	Not accepted
2	Enter 6 to 10 character	Accepted

3	Enter 11 to 14 character	Not Accepted

5. What is Integration Testing?

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit-tested, integration testing is performed.

Integration test approaches – There are four types of integration testing approaches. Those approaches are the following:

1. Big-Bang Integration Testing – It is the simplest integration testing approach, where all the modules are combined and the functionality is verified after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested.

Advantages:

- It is convenient for small systems.
- Simple and straightforward approach.
- Can be completed quickly.
- Does not require a lot of planning or coordination.
- May be suitable for small systems or projects with a low degree of interdependence between components.

Disadvantages:

- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.

- High-risk critical modules are not isolated and tested on priority since all modules are tested at once.
- Not Good for long projects.
- High risk of integration problems that are difficult to identify and diagnose.
- This can result in long and complex debugging and troubleshooting efforts.
- This can lead to system downtime and increased development costs.
- May not provide enough visibility into the interactions and data exchange between components.
- This can result in a lack of confidence in the system's stability and reliability.
- This can lead to decreased efficiency and productivity.
- This may result in a lack of confidence in the development team.
- This can lead to system failure and decreased user satisfaction.

2. Bottom-Up Integration Testing – In bottom-up testing, each module at lower levels are tested with higher modules until all modules are tested.

Advantages:

- In bottom-up testing, no stubs are required.
- A principal advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.
- It is easy to create the test conditions.
- Best for applications that uses bottom up design approach.
- It is Easy to observe the test results.

Disadvantages:

- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystems.
- As Far modules have been created, there is no working model can be represented.

3. Top-Down Integration Testing – Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet

integrated. In this integration testing, testing takes place from top to bottom. First, high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

Advantages:

- Separately debugged module.
- Few or no drivers needed.
- It is more stable and accurate at the aggregate level.
- Easier isolation of interface errors.
- In this, design defects can be found in the early stages.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.
- It is difficult to observe the test output.
- It is difficult to stub design.

4. Mixed Integration Testing – A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested.

Advantages:

- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.
- Parallel test can be performed in top and bottom layer tests.

Disadvantages:

- For mixed integration testing, it requires very high cost because one part has a Top-down approach while another part has a bottom-up approach.

- This integration testing cannot be used for smaller systems with huge interdependence between different modules.

Applications:

1. **Identify the components:** Identify the individual components of your application that need to be integrated. This could include the frontend, backend, database, and any third-party services.
2. **Create a test plan:** Develop a test plan that outlines the scenarios and test cases that need to be executed to validate the integration points between the different components. This could include testing data flow, communication protocols, and error handling.
3. **Set up test environment:** Set up a test environment that mirrors the production environment as closely as possible. This will help ensure that the results of your integration tests are accurate and reliable.
4. **Execute the tests:** Execute the tests outlined in your test plan, starting with the most critical and complex scenarios. Be sure to log any defects or issues that you encounter during testing.
5. **Analyze the results:** Analyze the results of your integration tests to identify any defects or issues that need to be addressed. This may involve working with developers to fix bugs or make changes to the application architecture.
6. **Repeat testing:** Once defects have been fixed, repeat the integration testing process to ensure that the changes have been successful, and that the application still works as expected.

6. What determines the level of risks?

When you think of levels of risk, the low, moderate, and high terms come to mind. However, in the real software testing scenario, the risk level is determined by two dimensions: probability and impact.

- **Probability**: It measures the likelihood of an event occurring, typically expressed as a percentage or qualitative scale. It answers the question: "How likely is it to happen?" It ranges from 0 to 100%. The probability can never be 0% because it indicates that there is no risk, and it can never be 100% as it indicates that it is no longer a risk but a certainty.
- **Impact**: Risk, by default, brings a negative impact to any project. It assesses the consequences or severity of an event, usually quantified in monetary, temporal, or qualitative terms. It answers the question: "What would be the extent of its effect?"
- By taking into account these two factors, you can calculate the level of the risk:
- ***Level of Risk in Software = Probability of Risk Occurring X Impact of the occurred risk***

Risk Assessment*:*

- Probability: You estimate a 20% probability of a security breach happening due to past incidents with the library.
- Impact: A security breach could result in data exposure and significant financial losses, potentially leading to a project delay.
- Risk Level Calculation:
- **Probability (20%) x Impact (High) = Risk Level (High)**

7. What Alpha Testing and Beta Testing?

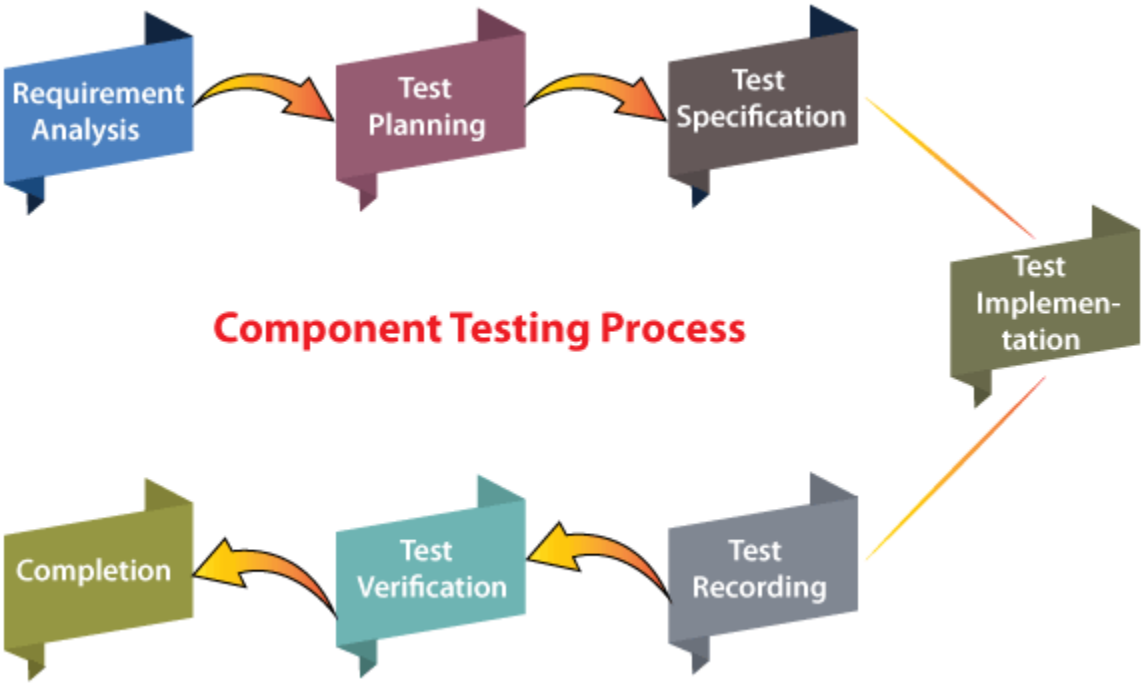
- Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance tests. It is the first stage of software testing, during which the internal development team tests the program before making it available to clients or people outside the company.
- Beta Testing is performed by real users of the software application in a real environment. Beta testing is one type of User Acceptance Testing. A pre-release version of the product is made available for testing to a chosen set of external users or customers during the second phase of software testing.

Parameters	Alpha Testing	Beta Testing
Involvment	Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black-box testing.
Performed by	Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Performed at	Alpha testing is performed at the developer's site.	Beta testing is performed at the end-user of the product.
Reliability and Security	Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Ensures	Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Requirement	Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.

Execution	Alpha testing may require a long execution cycle.	Beta testing requires only a few weeks of execution.
Issues	Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from the beta testing will be implemented in future versions of the product.
Test Cycles	Multiple test cycles are organized in alpha testing.	Only one or two test cycles are there in beta testing.

8. What is Component Testing?

- Component testing is a level of software testing that focuses on verifying the functionality and correctness of individual components or units within a system. It can be performed using white-box or black-box approaches, with the aim of ensuring that each component behaves as expected.



9. What is Functional and Non-Functional Testing?

- **Functional Testing:** It is a type of software testing in which the system is tested against the functional requirements and specifications. Functional testing ensures that the requirements or specifications are properly satisfied by the application. This type of testing is particularly concerned with the result of processing.
- **Non-functional Testing:** It is a type of software testing that is performed to verify the non-functional requirements of the application. It verifies whether the behavior of the system is as per the requirement or not. It tests all the aspects which are not tested in functional testing.
- **Below is the difference between functional and non-functional testing:**

Functional Testing	Non-functional Testing
It verifies the operations and actions of an application.	It verifies the behavior of an application.
It is based on requirements of customer.	It is based on expectations of customer.
It helps to enhance the behavior of the application.	It helps to improve the performance of the application.
Functional testing is easy to execute manually.	It is hard to execute non-functional testing manually.
It tests what the product does.	It describes how the product does.
Functional testing is based on the business requirement.	Non-functional testing is based on the performance requirement.

Examples:	Examples:
1. Unit Testing **2.** Smoke Testing **3.** Integration Testing **4.** Regression Testing	**1.** Performance Testing **2.** Load Testing **3.** Stress Testing **4.** Scalability Testing

10. What is GUI Testing?

Graphical User Interface Testing (GUI) Testing is the process for ensuring proper functionality of the graphical user interface (GUI) for a specific application.GUI testing generally evaluates a design of elements such as layout, colors and also fonts, font sizes, labels, text boxes, text formatting, captions, buttons, lists, icons, links, and content. GUI testing processes may be either manual or automatic and are often performed by third-party companies, rather than developers or end users.

Feature of Graphical User Interface Testing (GUI):

There are some feature of GUI which are given below:

- It provides a customizable test report.
- It is run tests in parallel or distributed on a Selenium Grid with built-in Selenium Webdriver.
- It allows you to test the functionality from a user's perspective.
- Sometimes the internal functions of the system work correctly but the user interface doesn't then GUI testing is good to have in addition to the other types.
- It provides reliable object identification, even for web elements with dynamic IDs.

Types of Graphical User Interface Testing (GUI) Testing:

There are two types of GUI testing which are given below: Analog Recording, and Object based recording. These are explained as following below.

1. **Analog Recording:**
   This is always what people associate with GUI testing tools by analog recording, the testing tool basically captures specific mouse clicks, keyboard presses, and other user actions and then simply stores them in a file for playback. For example, it might record that a user left-clicked at position X = 500 pixels, Y = 400 pixels or typed the word “Search” in a box and pressed the [ENTER] key on their keyboard.
2. **Object based Recording:**
   In object based recording, the testing tool is able to connect programmatically to the application being tested and “see” each of the individual user interface components (a button, a text box, a hyperlink) as separate entities and is able to perform operations (click, enter text) and read the state (is it enabled, what is the label text, what is the current value) reliably regardless of where that object is on the screen.

Challenges with Graphical User Interface Testing (GUI) Testing:
There are some challenges that occur during Graphical user interface testing. These are given below.

1. Technology Support
2. Stability of Objects
3. Instrumentation

11. What is Adhoc Testing?

Adhoc Testing : Adhoc testing is a type of software testing that is performed informally and randomly after the formal testing is completed to find any loophole in the system. For this reason, it is also known as Random or Monkey testing. Adhoc testing is not performed in a structured way so it is not based on any methodological approach. That’s why Adhoc testing is a type of Unstructured Software Testing.

Adhoc testing has –

- No Documentation.
- No Test cases.
- No Test Design.

As it is not based on any test cases or requires documentation or test design resolving issues that are identified at last becomes very difficult for developers. Sometimes very interesting and unexpected errors or uncommon errors are found which would never have been found in written test cases. This Adhoc testing is used in Acceptance testing.

Adhoc testing saves a lot of time and one great example of Adhoc testing can be when the client needs the product by today 6 PM but the product development will be completed at 4 PM the same day. So in hand only limited time i.e. 2 hours only, within that 2hrs the developer and tester team can test the system as a whole by taking some random inputs and can check for any errors.

Types of Adhoc Testing

Adhoc testing is divided into three types as follows.

1. **Buddy Testing –** Buddy testing is a type of Adhoc testing where two bodies will be involved one is from the Developer team and one from the tester team. So that after completing one module and after completing Unit testing the tester can test by giving random inputs and the developer can fix the issues too early based on the currently designed test cases.
2. **Pair Testing –** Pair testing is a type of Adhoc testing where two bodies from the testing team can be involved to test the same module. When one tester can perform the random test, another tester can maintain the record of findings. So, when two testers get paired, they exchange their ideas, opinions, and knowledge so good testing is performed on the module.
3. **Monkey Testing –** Monkey testing is a type of Adhoc testing in which the system is tested based on random inputs without any test cases the behavior of the system is tracked, and all the functionalities of the system are working or not is monitored. As the randomness approach is followed there is no constraint on inputs, so it is called Monkey testing.

Characteristics of Adhoc Testing

- Adhoc testing is performed randomly.
- Based on no documentation, no test cases, and no test designs.
- It is done after formal testing.
- It follows an unstructured way of testing.
- It takes comparatively less time than other testing techniques.
- It is good for finding bugs and inconsistencies that are mentioned in test cases.

When to conduct Adhoc testing

- When there is limited time in hand to test the system.
- When there are no clear test cases to test the product.
- When formal testing is completed.
- When the development is mostly complete.

When not to conduct Adhoc testing

- When an error exists in the test cases.
- When Beta testing is being carried out.

Advantages of Adhoc testing

- The errors that cannot be identified with written test cases can be identified by Adhoc testing.
- It can be performed within a very limited time.
- Helps to create unique test cases.
- This test helps to build a strong product that is less prone to future problems.
- This testing can be performed at any time during Software Development Life Cycle Process (SDLC)

Disadvantages of Adhoc testing

- Sometimes resolving errors based on identified issues is difficult as no written test cases and documents are there.
- Needs good knowledge of the product as well as testing concepts to perfectly identify the issues in any model.

- It does not provide any assurance that the error will be identified.
- Finding one error may take some uncertain period.

Five practices to follow to conduct Adhoc testing

1. Good Software Knowledge.
2. Find Out Error-Prone Areas.
3. Prioritize Test Areas.
4. Roughly Plan The Test Plan.
5. Use of the right kind of tools.

12. What is Load and Stress Testing?

- **Load testing** is a type of Performance Testing that determines the performance of a system, software product, or software application under real-life-based load conditions. This article focuses on discussing load testing in detail.

Load Testing Techniques:

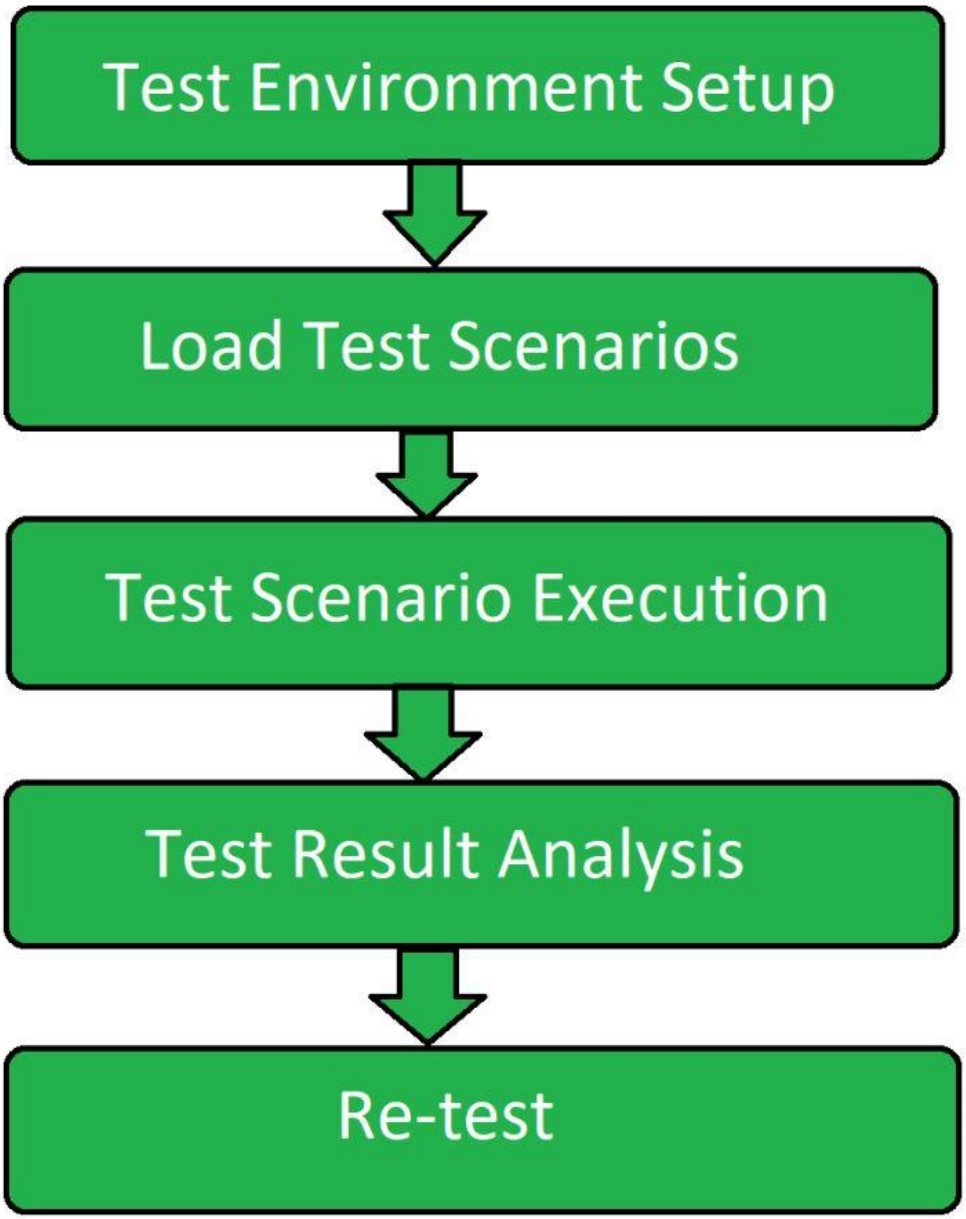
- **Stress testing:** Testing the system's ability to handle a high load above normal usage levels
- **Spike testing:** Testing the system's ability to handle sudden spikes in traffic
- **Soak testing**: Testing the system's ability to handle a sustained load over a prolonged period of time
- **Tools for Performance Testing:** Make use of specialized load testing tools like Locust, Gatling, JMeter, LoadRunner, and Apache Benchmark. These tools assist in gathering performance measurements and simulating a large number of users.
- **Specify the Test Objectives:** Clearly state what your load test's goals are. Recognize the required response times, transaction volumes and expected user behavior.
- **Determine Crucial Situations:** Determine the essential user scenarios that correspond to common usage patterns. A variety of actions, including user

logins, searches, form submissions and other significant interactions, should be covered by these scenarios.

Objectives of Load Testing:

- **Evaluation of Scalability:** Assess the system's ability to handle growing user and transaction demands. Find the point at which the system begins to function badly.
- **Planning for Capacity:** Describe the system's ability to accommodate anticipated future increases in the number of users, transactions and volume of data. Making well-informed decisions regarding infrastructure upgrades is made easier by this.
- **Determine bottlenecks:** Identify and localize bottlenecks in the application or infrastructure's performance. Finding the places where the system's performance can suffer under load is part of this.
- **Analysis of Response Time:** For crucial transactions and user interactions, track and evaluate response times. Make that the system responds to changes in load with reasonable response times.
- **Finding Memory Leaks**: Find and fix memory leaks that may eventually cause a decline in performance. Make sure the programme doesn't use up too many resources when it's running.

- **Load Testing Process:**



1. **Test Environment Setup:** Firstly create a dedicated test environment setup for performing the load testing. It ensures that testing would be done in a proper way.
2. **Load Test Scenario:** In second step load test scenarios are created. Then load testing transactions are determined for an application and data is prepared for each transaction.
3. **Test Scenario Execution:** Load test scenarios that were created in previous step are now executed. Different measurements and metrices are gathered to collect the information.

4. **Test Result Analysis:** Results of the testing performed is analyzed and various recommendations are made.
5. **Re-test:** If the test is failed then the test is performed again in order to get the result in correct way.

Metrics of Load Testing:

1**. Average Response Time -** It tells the average time taken to respond to the request generated by the clients or customers or users. It also shows the speed of the application depending upon the time taken to respond to the all requests generated.

2. **Error Rate -** The Error Rate is mentioned in terms of percentage denotes the number of errors occurred during the requests to the total number of requests. These errors are usually raised when the application is no longer handling the request at the given time or for some other technical problems. It makes the application less efficient when the error rate keeps on increasing.

3. **Throughput -** This metric is used in knowing the range of bandwidth consumed during the load scripts or tests and it is also used in knowing the amount of data which is being used for checking the request that flows between the user server and application main server. It is measured in kilobytes per second.

4. **Requests Per Second -** It tells that how many requests are being generated to the application server per second. The requests could be anything like requesting of images, documents, web pages, articles or any other resources.

5. **Concurrent Users -** This metric is used to take the count of the users who are actively present at the particular time or at any time. It just keeps track of count those who are visiting the application at any time without raising any request in the application. From this, we can easily know that at which time the high number of users are visiting the application or website.

6. Peak Response Time - Peak Response Time measures the time taken to handle the request. It also helps in finding the duration of the peak time(longest time) at which the request and response cycle is handled and finding that which resource is taking longer time to respond the request.

Load Testing Tools:

1. Apache Jmeter
2. WebLoad
3. NeoLoad
4. LoadNinja
5. HP Performance Tester
6. LoadUI Pro
7. LoadView

Difference between Load and Stress testing:

Load Testing	Stress Testing
Load testing identifies the bottlenecks in the system under various workloads and checks how the system reacts when the load is gradually increased	Stress Testing determines the breaking point of the system to reveal the maximum point after which it breaks.
To recognize the upper limit of the system, set SLA of the app and check how the system can handle a heavy load.	To check out how the system behaves under extreme loads and how it recovers from failure.
Generating increased load on a web application is the main aim of load testing.	Stress testing aims to ensure that under a sudden high load for a considerable duration the servers don't crash.
The attributes which are checked in a load test are peak performance, server quantity and response time.	This kind of testing checks stability response time, etc.
In load testing load limit is a threshold of a break.	In stress testing load limit is above the threshold of a break.

13. What is Stress Testing?

Stress Testing is a software testing technique that determines the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for critical software but is used for all types of software.

Characteristics of Stress Testing:

1. **Identification of Risk:** Stress testing's main objective is to locate and evaluate a system's possible hazards and weaknesses.
2. **Quantitative and Qualitative Analysis:** While numerical data are crucial, it's also critical to comprehend the qualitative characteristics of the system's response and potential weak points.
3. **Variable Parameters:** Stress testing include changing variables including interest rates, market conditions, transaction volumes and outside influences that could have an impact on the system.
4. **Cross-Functional Involvement:** Many departments within an organization must work together and participate in stress testing. This cross-functional strategy makes sure that the stress testing procedure benefits from a variety of viewpoints and specialties.
5. **Open and Honest Communication:** Stress testing necessitates open and honest communication regarding the goal, approach, and outcomes of the testing procedure.

Purpose of Stress Testing:

- **Analyze the behavior of the application after failure:** The purpose of stress testing is to analyze the behavior of the application after failure and the software should display the appropriate error messages while it is under extreme conditions.
- **System recovers after failure:** Stress testing aims to make sure that there are plans for recovering the system to the working state so that the system recovers after failure.
- **Uncover Hardware issues:** Stress testing helps to uncover hardware issues and data corruption issues.

- **Uncover Security Weakness:** Stress testing helps to uncover the security vulnerabilities that may enter into the system during the constant peak load and compromise the system.
- **Ensures data integrity:** Stress testing helps to determine the application's data integrity throughout the extreme load, which means that the data should be in a dependable state even after a failure.

Stress Testing Process:

The stress testing process is divided into 5 steps:

1. **Planning the stress test:** This step involves gathering the system data, analyzing the system, and defining the stress test goals.
2. **Create Automation Scripts:** This step involves creating the stress testing automation scripts and generating the test data for the stress test scenarios.
3. **Script Execution:** This step involves running the stress test automation scripts and storing the stress test results.
4. **Result Analysis:** This phase involves analyzing stress test results and identifying the bottlenecks.
5. **Tweaking and Optimization:** This step involves fine-tuning the system and optimizing the code with the goal meet the desired benchmarks.
6. Pages Per Second: Number of pages requested per second and number of pages loaded per second.
7. Pages Retrieved: Average time is taken to retrieve all information from a particular page.
8. Byte Retrieved: Average time is taken to retrieve the first byte of information from the page.
9. Transaction Response Time: Average time is taken to load or perform transactions between the applications.
10. Transactions per Second: It takes count of the number of transactions loaded per second successfully and it also counts the number of failures that occurred.

11. Failure of Connection: It takes count of the number of times that the client faced connection failure in their system.
12. Failure of System Attempts: It takes count of the number of failed attempts in the system.
13. Rounds: It takes count of the number of test or script conditions executed by the clients successfully and it keeps track of the number of rounds failed.

Stress Testing Tools:

1. **Jmeter:** Apache JMeter is a stress testing tool is an open-source, pure Java-based software that is used to stress test websites. It is an Apache project and can be used for load testing for analyzing and measuring the performance of a variety of services.
2. **LoadNinja:** LoadNinja is a stress testing tool developed by SmartBear that enables users to develop codeless load tests, substitutes load emulators with actual browsers, and helps to achieve high speed and efficiency with browser-based metrics.
3. **WebLoad:** WebLoad is a stress testing tool that combines performance, stability, and integrity as a single process for the verification of mobile and web applications.
4. **Neoload:** Neoload is a powerful performance testing tool that simulates large numbers of users and analyzes the server's behavior. It is designed for both mobile and web applications. Neoload supports API testing and integrates with different CI/ CD applications.
5. **SmartMeter:** SmartMeter is a user-friendly tool that helps to create simple tests without coding. It has a graphical user interface and has no necessary plugins. This tool automatically generates advanced test reports with complete and detailed test results.

Metrics of Stress Testing:

Metrics are used to evaluate the performance of the stress and it is usually carried out at the end of the stress scripts or tests. Some of the metrics are given below.

1. **Pages Per Second:** Number of pages requested per second and number of pages loaded per second.
2. **Pages Retrieved:** Average time is taken to retrieve all information from a particular page.
3. **Byte Retrieved:** Average time is taken to retrieve the first byte of information from the page.
4. **Transaction Response Time:** Average time is taken to load or perform transactions between the applications.
5. **Transactions per Second:** It takes count of the number of transactions loaded per second successfully and it also counts the number of failures that occurred.
6. **Failure of Connection:** It takes count of the number of times that the client faced connection failure in their system.
7. **Failure of System Attempts:** It takes count of the number of failed attempts in the system.
8. **Rounds:** It takes count of the number of test or script conditions executed by the clients successfully and it keeps track of the number of rounds failed.

14. What is White Box Testing?

White box testing techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing.

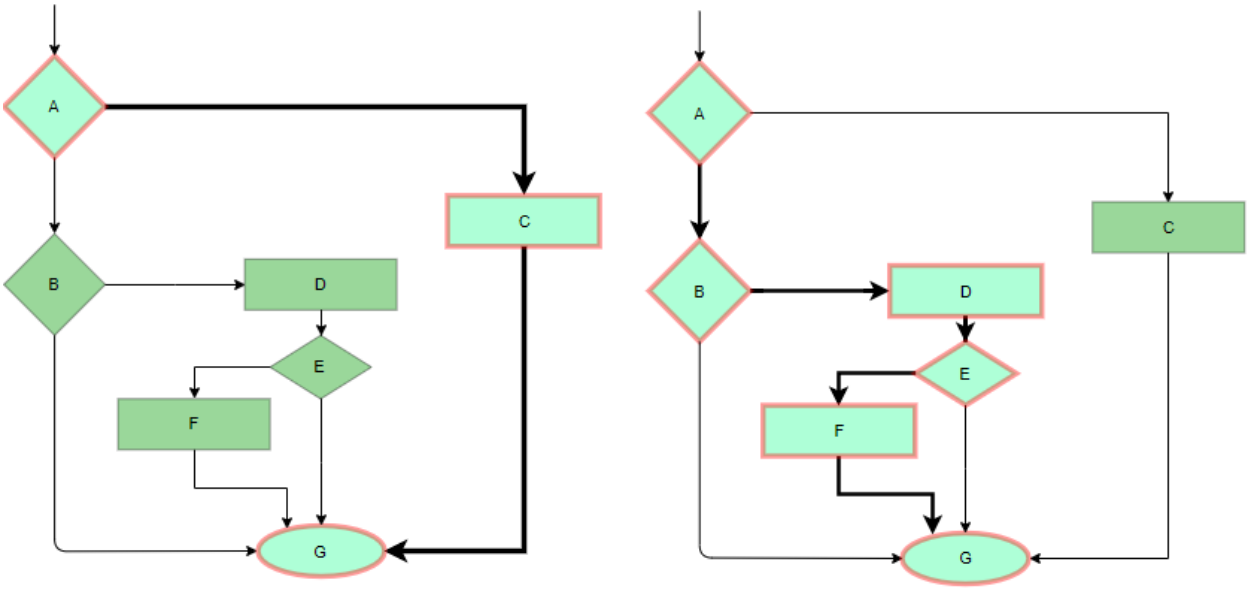
White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.

Process of White Box Testing

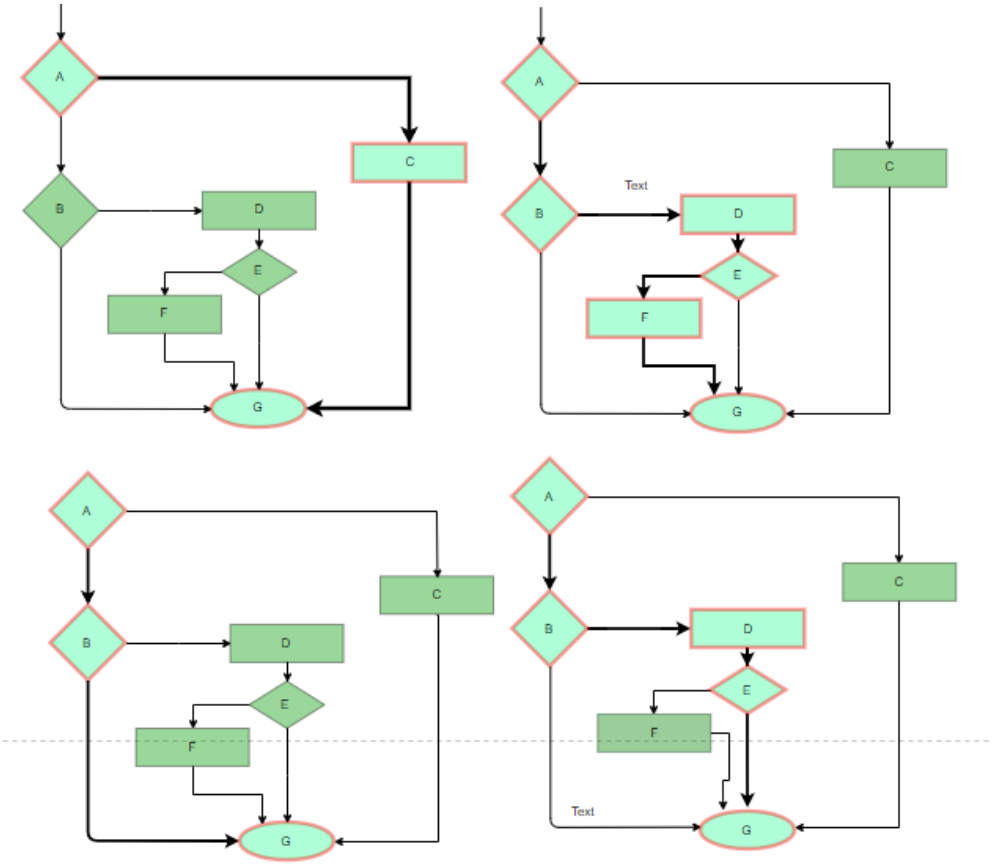
1. **Input:** Requirements, Functional specifications, design documents, source code.
2. **Processing:** Performing risk analysis to guide through the entire process.
3. **Proper test planning:** Designing test cases to cover the entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
4. **Output:** Preparing final report of the entire testing process.

Testing Techniques

1. Statement Coverage - In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.



2. Branch Coverage: In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.



3. Condition Coverage

In this technique, all individual conditions must be covered as shown in the following example:

- READ X, Y
- IF(X == 0 || Y == 0)
- PRINT '0'
- #TC1 – X = 0, Y = 55
- #TC2 – X = 5, Y = 0

4. Multiple Condition Coverage

In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let's consider the following example:

- READ X, Y
- IF(X == 0 || Y == 0)
- PRINT ‘0’
- #TC1: X = 0, Y = 0
- #TC2: X = 0, Y = 5
- #TC3: X = 55, Y = 0
- #TC4: X = 55, Y = 5

5. Basis Path Testing

In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path. **Steps:**

- Make the corresponding control flow graph
- Calculate the cyclomatic complexity
- Find the independent paths
- Design test cases corresponding to each independent path
- V(G) = P + 1, where P is the number of predicate nodes in the flow graph
- V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes
- V(G) = Number of non-overlapping regions in the graph
- #P1: 1 – 2 – 4 – 7 – 8
- #P2: 1 – 2 – 3 – 5 – 7 – 8
- #P3: 1 – 2 – 3 – 6 – 7 – 8
- #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

6. Loop Testing

Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.

- **Simple loops:** For simple loops of size n, test cases are designed that:

15. Skip the loop entirely
16. Only one pass through the loop
17. 2 passes
18. m passes, where m < n
19. n-1 ans n+1 passes

- **Nested loops:** For nested loops, all the loops are set to their minimum count, and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
- **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each. If they're not independent, treat them like nesting.

White Testing is performed in 2 Steps

1. Tester should understand the code well
2. Tester should write some code for test cases and execute them

Tools required for White box testing:

- PyUnit
- Sqlmap
- Nmap
- Parasoft Jtest
- Nunit
- VeraUnit
- CppUnit
- Bugzilla
- Fiddler
- JSUnit.net
- OpenGrok
- Wireshark
- HP Fortify

- CSUnit

Features of White box Testing

1. **Code coverage analysis:** White box testing helps to analyze the code coverage of an application, which helps to identify the areas of the code that are not being tested.
2. **Access to the source code:** White box testing requires access to the application's source code, which makes it possible to test individual functions, methods, and modules.
3. **Knowledge of programming languages:** Testers performing white box testing must have knowledge of programming languages like Java, C++, Python, and PHP to understand the code structure and write tests.
4. **Identifying logical errors:** White box testing helps to identify logical errors in the code, such as infinite loops or incorrect conditional statements.
5. **Integration testing:** White box testing is useful for integration testing, as it allows testers to verify that the different components of an application are working together as expected.
6. **Unit testing:** White box testing is also used for unit testing, which involves testing individual units of code to ensure that they are working correctly.
7. **Optimization of code:** White box testing can help to optimize the code by identifying any performance issues, redundant code, or other areas that can be improved.
8. **Security testing:** White box testing can also be used for security testing, as it allows testers to identify any vulnerabilities in the application's code.
9. **Verification of Design:** It verifies that the software's internal design is implemented in accordance with the designated design documents.
10. **Check for Accurate Code:** It verifies that the code operates in accordance with the guidelines and specifications.
11. **Identifying Coding Mistakes:** It finds and fix programming flaws in your code, including syntactic and logical errors.
12. **Path Examination:** It ensures that each possible path of code execution is explored and test various iterations of the code.

13. **Determining the Dead Code:** It finds and remove any code that isn't used when the programme is running normally (dead code).

Advantages of Whitebox Testing

1. **Thorough Testing**: White box testing is thorough as the entire code and structures are tested.
2. **Code Optimization:** It results in the optimization of code removing errors and helps in removing extra lines of code.
3. **Early Detection of Defects:** It can start at an earlier stage as it doesn't require any interface as in the case of black box testing.
4. **Integration with SDLC:** White box testing can be easily started in Software Development Life Cycle.
5. **Detection of Complex Defects:** Testers can identify defects that cannot be detected through other testing techniques.
6. **Comprehensive Test Cases:** Testers can create more comprehensive and effective test cases that cover all code paths.
7. Testers can ensure that the code meets coding standards and is optimized for performance.

Disadvantages of White box Testing

1. **Programming Knowledge and Source Code Access:** Testers need to have programming knowledge and access to the source code to perform tests.
2. **Overemphasis on Internal Workings:** Testers may focus too much on the internal workings of the software and may miss external issues.
3. **Bias in Testing:** Testers may have a biased view of the software since they are familiar with its internal workings.
4. **Test Case Overhead:** Redesigning code and rewriting code needs test cases to be written again.
5. **Dependency on Tester Expertise:** Testers are required to have in-depth knowledge of the code and programming language as opposed to black-box testing.

6. **Inability to Detect Missing Functionalities:** Missing functionalities cannot be detected as the code that exists is tested.
7. **Increased Production Errors:** High chances of errors in production.

20. What is Black Box Testing?

Black-box testing is a type of software testing in which the tester is not concerned with the internal knowledge or implementation details of the software but rather focuses on validating the functionality based on the provided specifications or requirements.

Types Of Black Box Testing

The following are the several categories of black box testing:

1. **Functional Testing**
2. **Regression Testing**
3. **Nonfunctional Testing (NFT)**

Tools Used for Black Box Testing:

1. **Appium**
2. **Selenium**
3. **Microsoft Coded UI**
4. **Applitools**
5. **HP QTP.**

What can be identified by Black Box Testing

1. Discovers missing functions, incorrect function & interface errors
2. Discover the errors faced in accessing the database
3. Discovers the errors that occur while initiating & terminating any functions.
4. Discovers the errors in performance or behaviour of software.

Features of black box testing

1. **Independent testing:** Black box testing is performed by testers who are not involved in the development of the application, which helps to ensure that testing is unbiased and impartial.
2. **Testing from a user's perspective:** Black box testing is conducted from the perspective of an end user, which helps to ensure that the application meets user requirements and is easy to use.
3. **No knowledge of internal code:** Testers performing black box testing do not have access to the application's internal code, which allows them to focus on testing the application's external behaviour and functionality.
4. **Requirements-based testing:** Black box testing is typically based on the application's requirements, which helps to ensure that the application meets the required specifications.
5. **Different testing techniques:** Black box testing can be performed using various testing techniques, such as functional testing, usability testing, acceptance testing, and regression testing.
6. **Easy to automate:** Black box testing is easy to automate using various automation tools, which helps to reduce the overall testing time and effort.
7. **Scalability:** Black box testing can be scaled up or down depending on the size and complexity of the application being tested.
8. **Limited knowledge of application:** Testers performing black box testing have limited knowledge of the application being tested, which helps to ensure that testing is more representative of how the end users will interact with the application.

21. Mention the categories of Defects.

- **Types of Defects:** Following are some of the basic types of defects in the software development:

1. **Arithmetic Defects:** It include the defects made by the developer in some arithmetic expression or mistake in finding solution of such arithmetic expression. This type of defects are basically made by the programmer due

to access work or less knowledge. Code congestion may also lead to the arithmetic defects as programmer is unable to properly watch the written code.

2. **Logical Defects:** Logical defects are mistakes done regarding the implementation of the code. When the programmer doesn't understand the problem clearly or thinks in a wrong way then such types of defects happen. Also while implementing the code if the programmer doesn't take care of the corner cases then logical defects happen. It is basically related to the core of the software.
3. **Syntax Defects:** Syntax defects means mistake in the writing style of the code. It also focuses on the small mistake made by developer while writing the code. Often the developers do the syntax defects as there might be some small symbols escaped. For example, while writing a code in C++ there is possibility that a semicolon(;) is escaped.
4. **Multithreading Defects:** Multithreading means running or executing the multiple tasks at the same time. Hence in multithreading process there is possibility of the complex debugging. In multithreading processes sometimes there is condition of the deadlock and the starvation is created that may lead to system's failure.
5. **Interface Defects:** Interface defects means the defects in the interaction of the software and the users. The system may suffer different kinds of the interface testing in the forms of the complicated interface, unclear interface or the platform based interface.
6. **Performance Defects:** Performance defects are the defects when the system or the software application is unable to meet the desired and the expected results. When the system or the software application doesn't fulfill the users's requirements then that is the performance defects. It also includes the response of the system with the varying load on the system.
7. **software error:** A software error occurs during the development of the software. This error can be in the form of a grammatical error, a logical error where the outcome of a sequence of executions will not result in what was intended or a misinterpretation of the user requirements in the

actual written code. It may be in the form of user documentation not matching the software applications operation. An error may or may not be detected during the coding or testing of the program before it is released to a customer.

8. **Software Fault:** A software fault occurs as a result of an error that remains in the executing program. Not all faults however are detected and the software may continue executing without any obvious problems. There are cases where software faults go undetected for many years of a programs existence.
9. **Software Failure:** A software failure is a fault that results in a detectable problem; hence it is referred to as a failure. A failure would cause the application to malfunction in an obvious manner that warrants the attention of system maintenance.
10. **Boundary and Range Defects:** These defects relate to issues where the software does not handle inputs or data outside of specified boundaries or ranges correctly. For example, not handling edge cases or boundary values appropriately.
11. **Data Validation Defects:** It involves failing to validate user inputs, leading to potential security vulnerabilities or incorrect data handling.
12. **Deployment Defects:** It includes setup mistakes, installation issues and problems that stop the software from functioning properly in a real-world setting.
13. **Integration Defects:** When different software modules or components do not function well together, integration errors result. Problems with data synchronization, communication, and functionality may result from this.
14. **Documentation Defects:** These refer to errors or issues in the accompanying documentation, including user manuals, help files or API documentation. Incomplete or inaccurate documentation can lead to user confusion.
15. **Data Defects:** These can result in data corruption or loss, inconsistent data storage or issues with data retrieval. For example, data validation errors could lead to incorrect data being stored.

22. What is the Purpose of exit criteria?

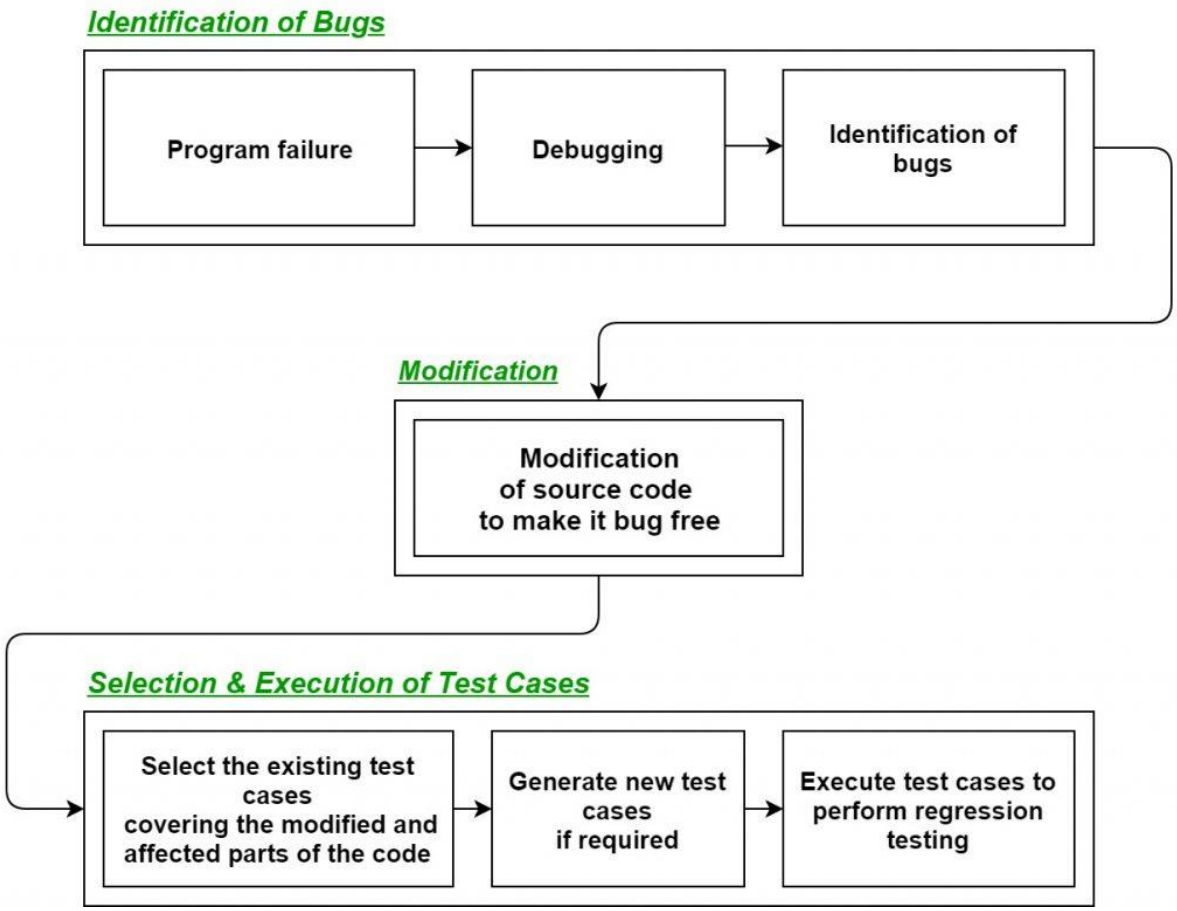
- Exit criterion is used to determine whether a given test activity has been completed or NOT. Exit criteria can be defined for all of the test activities right from planning, specification and execution.
- Exit criterion should be part of test plan and decided in the planning stage.

Examples of Exit Criteria:

- Verify if All tests planned have been run.
- Verify if the level of requirement coverage has been met.
- Verify if there are NO Critical or high severity defects that are left outstanding.
- Verify if all high risk areas are completely tested.
- Verify if software development activities are completed within the projected cost.
- Verify if software development activities are completed within the projected timelines.

23. When should “Regression Testing” be performed ?

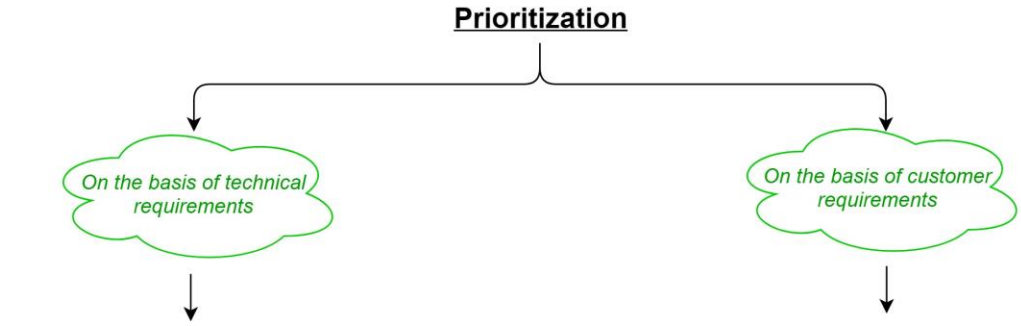
- When new functionality is added to the system and the code has been modified to absorb and integrate that functionality with the existing code.
- When some defect has been identified in the software and the code is debugged to fix it.
- When the code is modified to optimize its working.



Techniques for the selection of Test cases for Regression Testing

- **Select all test cases:** In this technique, all the test cases are selected from the already existing test suite. It is the simplest and safest technique but not very efficient.
- **Select test cases randomly:** In this technique, test cases are selected randomly from the existing test suite, but it is only useful if all the test cases are equally good in their fault detection capability which is very rare. Hence, it is not used in most of the cases.
- **Select modification traversing test cases:** In this technique, only those test cases are selected that cover and test the modified portions of the source code and the parts that are affected by these modifications.
- **Select higher priority test cases:** In this technique, priority codes are assigned to each test case of the test suite based upon their bug detection capability, customer requirements, etc. After assigning the priority codes, test cases with the highest priorities are selected for the process of regression testing. The

test case with the highest priority has the highest rank. For example, a test case with priority code 2 is less important than a test case with priority code 1.



Priority Code 1: Essential Test Case

Priority Code 2: Important Test Case

Priority Code 3: Execute, if resources permits

Priority Code 4: Not important Test Case

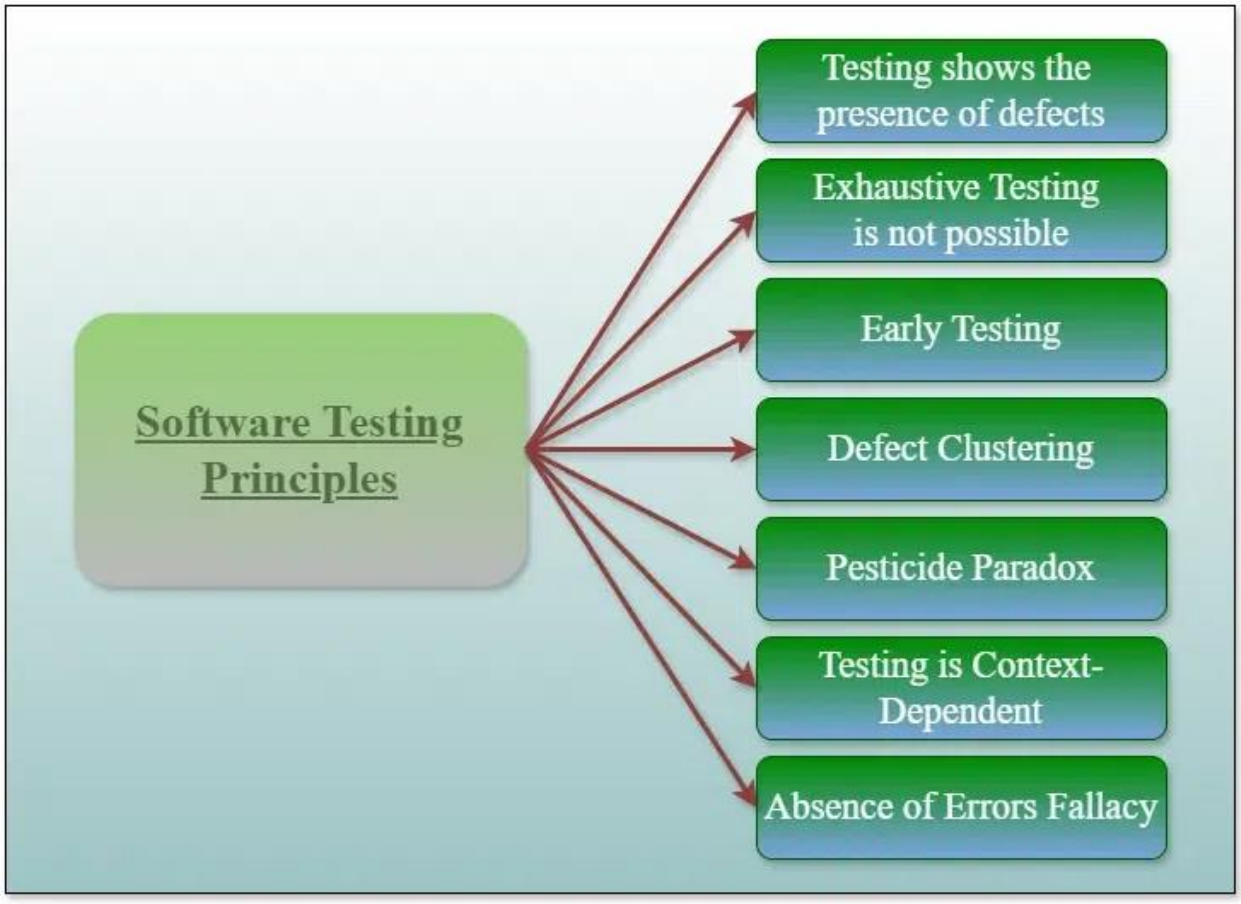
Priority Code 5: Redundant Test Case

Priority Code 1: Important for the customer

Priority Code 2: Required to increase the customer satisfaction

Priority Code 3: Helps to increase the market share of the product

24. What is 7 key Principles in Software Testing ?



1. Testing shows the Presence of Defects:

The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn't talk about the absence of defects. Software testing can ensure that defects are present but it can not prove that software is defect-free. Even multiple tests can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects.

2. Exhaustive Testing is not Possible:

It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will

produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.

3. Early Testing:

To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

4. Defect Clustering:

In a project, a small number of modules can contain most of the defects. The Pareto Principle for software testing states that 80% of software defects come from 20% of modules.

5. Pesticide Paradox:

Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

6. Testing is Context-Dependent:

The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.

7. Absence of Errors Fallacy:

If a built software is 99% bug-free but does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

Types of Software Testing

1. **Unit Testing**
2. **Integration Testing**
3. **Regression Testing**
4. **Smoke Testing**
5. **System Testing**
6. **Alpha Testing**
7. **Beta Testing**
8. **Performance Testing**

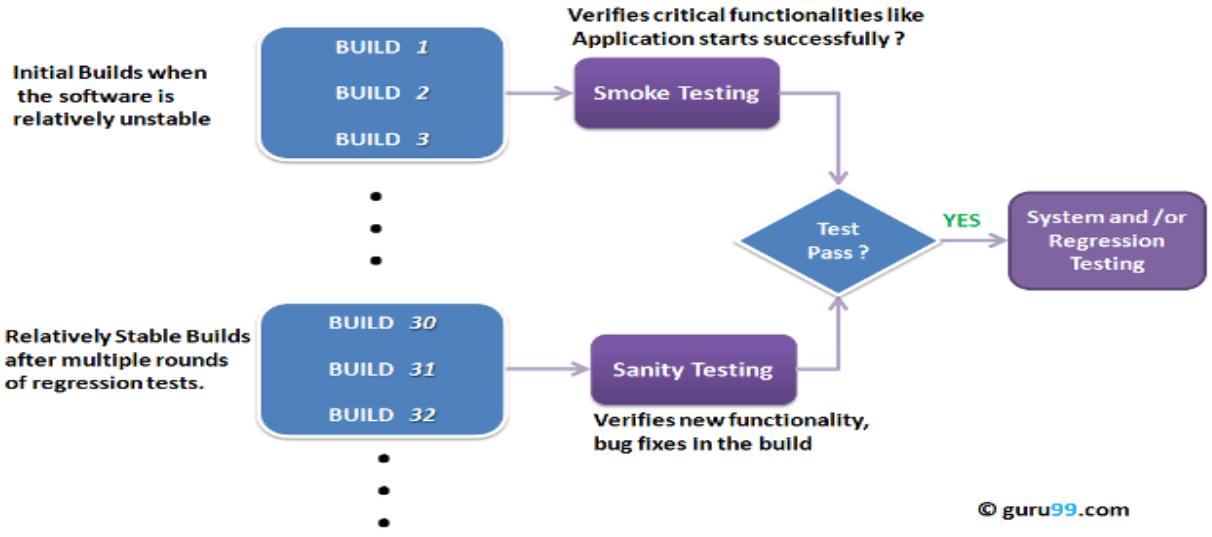
25. What is Difference between QA and QC in Software Testing ?

Parameters	Quality Assurance (QA)	Quality Control (QC)
Objective	It focuses on providing assurance that the quality requested will be achieved.	It focuses on fulfilling the quality requested.
Technique	It is the technique of managing quality.	It is the technique to verify quality.
Involved in which phase?	It is involved during the development phase.	It is not included during the development phase.
Program execution is included?	It does not include the execution of the program.	It always includes the execution of the program.
Type of tool	It is a managerial tool.	It is a corrective tool.

Process/ Product-oriented	It is process oriented.	It is product oriented.
Aim	The aim of quality assurance is to prevent defects.	The aim of quality control is to identify and improve the defects.
Order of execution	It is performed before Quality Control.	It is performed after the Quality Assurance activity is done.
Technique type	It is a preventive technique.	It is a corrective technique.
Measure type	It is a proactive measure.	It is a reactive measure.
SDLC/ STLC?	It is responsible for the entire software development life cycle.	It is responsible for the software testing life cycle.
Activity level	QA is a low-level activity that identifies an error and mistakes that QC cannot.	It is a high-level activity that identifies an error that QA cannot.
Focus	Pays main focus is on the intermediate process.	Its primary focus is on final products.
Team	All team members of the project are involved.	Generally, the testing team of the project is involved.
Aim	It aims to prevent defects in the system.	It aims to identify defects or bugs in the system.

Time consumption	It is a less time-consuming activity.	It is a more time-consuming activity.
Which statistical technique was applied?	Statistical Process Control (SPC) statistical technique is applied to Quality Assurance.	Statistical Quality Control (SQC) statistical technique is applied to Quality Control.
Example	Verification	Validation

26. Difference between Smoke and Sanity Testing ?



Following is the difference between Sanity and Smoke testing:

Smoke Testing	Sanity Testing
Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine	Sanity Testing is done to check the new functionality/bugs have been fixed

The objective of this testing is to verify the “stability” of the system in order to proceed with more rigorous testing	The objective of the testing is to verify the “rationality” of the system in order to proceed with more rigorous testing
This testing is performed by the developers or testers	Sanity testing in software testing is usually performed by testers
Smoke testing is usually documented or scripted	Sanity testing is usually not documented and is unscripted
Smoke testing is a subset of Acceptance testing	Sanity testing is a subset of Regression Testing
Smoke testing exercises the entire system from end to end	Sanity testing exercises only the particular component of the entire system
Smoke testing is like General Health Check Up	Sanity Testing is like specialized health check up

27. Difference between Verification and Validation ?

Verification	Validation
It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.
It does *not* include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.

It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.
Verification refers to the set of activities that ensure software correctly implements the specific function.	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements.
After a valid and complete specification the verification starts.	Validation begins as soon as project starts.
Verification is for prevention of errors.	Validation is for detection of errors.
Verification is also termed as white box testing or static testing as work product goes through reviews.	Validation can be termed as black box testing or dynamic testing as work product is executed.

Verification finds about 50 to 60% of the defects.	Validation finds about 20 to 30% of the defects.
Verification is based on the opinion of reviewer and may change from person to person.	Validation is based on the fact and is often stable.
Verification is about process, standard and guideline.	Validation is about the product.

28. Explain types of Performance Testing ?

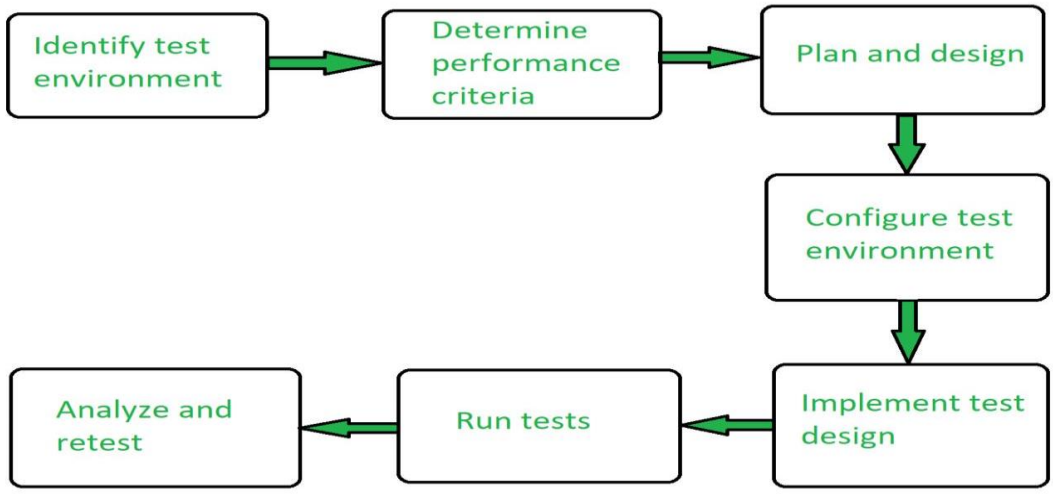
Types of Performance Testing:

1. ***Load testing*:**
   It checks the product's ability to perform under anticipated user loads. The objective is to identify performance congestion before the software product is launched in the market.
2. ***Stress testing*:**
   It involves testing a product under extreme workloads to see whether it handles high traffic or not. The objective is to identify the breaking point of a software product.
3. ***Endurance testing*:**
   It is performed to ensure the software can handle the expected load over a long period.
4. ***Spike testing*:**
   It tests the product's reaction to sudden large spikes in the load generated by users.
5. ***Volume testing*:**
   In volume testing, large number of data is saved in a database and the overall software system's behaviour is observed. The objective is to check the product's performance under varying database volumes.
6. ***Scalability testing*:**
   In scalability testing, the software application's effectiveness is determined by scaling up to support an increase in user load. It helps in planning capacity additions to your software system.

Performance Testing Tools:

Jmeter
Open STA
Load Runner
Web Load

Performance Testing Process:



29. What is Error, Defect, Bug and Failure ?

Bug vs Defect vs Error vs Fault vs Failure
Some of the vital differences between bug, defect, fault, error, and failure are listed in the below table:

Basis	Bug	Defect	Fault	Error	Failure
Definition	A bug refers to defects which means that the software product or	A Defect is a deviation between the actual and expected output	A Fault is a state that causes the software to fail and therefore it	An Error is a mistake made in the code due to which compilation	Failure is the accumulation of several defects that ultimately lead to Software failure and

	the application is not working as per the adhered requirements set		does not achieve its necessary function.	or execution fails,	results in the loss of information in critical modules thereby making the system unresponsive.
Raised by	Test Engineers	The defect is identified by The Testers And is resolved by developers in the development phase of SDLC.	Human mistakes lead to fault.	Developers and automation test engineers	The failure is found by the test engineer during the development cycle of SDLC
Different types	• Logical bugs • Algorithmic bugs • Resource bugs	Defects are classified as follows: **Based on Priority:** • High • Medium • Low **Based on Severity:** • Critical • Major • Minor	• Business Logic Faults • Functional and Logical Faults • Graphical User Interface (GUI) Faults	• Syntactic Error • UI screen error • Error handling error • Flow control error • Calculation error	NA

		• Trivial	• Performance Faults • Security Faults • Hardware Faults	• Hardware error	
Reasons behind	• Missing Logic • Erroneous Logic • Redundant codes	• Receiving & providing incorrect input • Coding/Logical Error leading to the breakdown of software	• Wrong design of the data definition processes. • An irregularity in Logic or gaps in the software leads to the non-functioning of the software.	• Error in code. • Inability to compile/ execute a program • Ambiguity in code logic • Misunderstanding of requirements • Faulty design and architecture • Logical error	• Environment variables • System Errors • Human Error
Way to	• Implementing Test-	• Implementing	• Peer review	• Conduct peer	• Confirmation of Re-testing

prevent the reasons	driven development. • Adjusting enhanced development practices and evaluation of cleanliness of the code.	Out-of-the-box programming methods. • Proper usage of primary and correct software coding practices.	of the Test documents and requirements. • Verifying the correctness of software design and coding.	reviews and code-reviews • Need for validation of bug fixes and enhancing the overall quality of the software.	the process end to end, • Carefully review the requirements as well as the specifications. • Categorizing and evaluating the errors and issues

30. What Difference between Priority and Severity ?

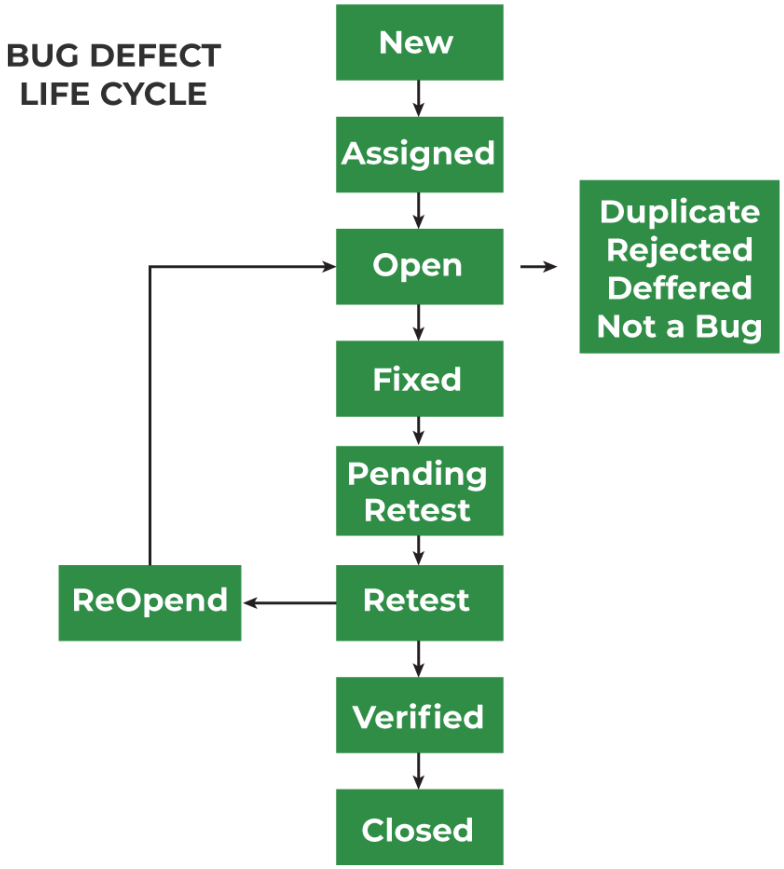
Severity vs Priority

Below are the differences between severity and priority:

Features	Severity	Priority
Definition	Severity is a parameter to denote the impact of a particular defect on the software.	Priority is a parameter to decide the order in which defects should be fixed.
Purpose	Severity means how severe the defect is affecting the functionality.	Priority means how fast the defect has to be fixed.

Relation	Severity is related to the quality standard.	Priority is related to scheduling to resolve the problem.
Categories	Severity is divided into 4 categories: • Critical • Major • Medium • Low	Priority is divided into 3 categories: • Low • Medium • High
Who decides defects?	The testing engineer decides the severity level of the defect.	The product manager decides the priorities of defects.
Value	Its value is objective.	Its value is subjective.
Value change	Its value doesn't change from time to time.	Its value changes from time to time.
Association	It is associated with functionality or standards.	It is associated with scheduling.
Indication	It indicates the seriousness of the bug in the product functionality.	It indicates how soon the bug should be fixed.
Driving factor	It is driven by functionality	It is driven by business value.
Based On	It is based on the technical aspect of the product.	It is based on the customer's requirements.

31. What is Bug Life Cycle ?

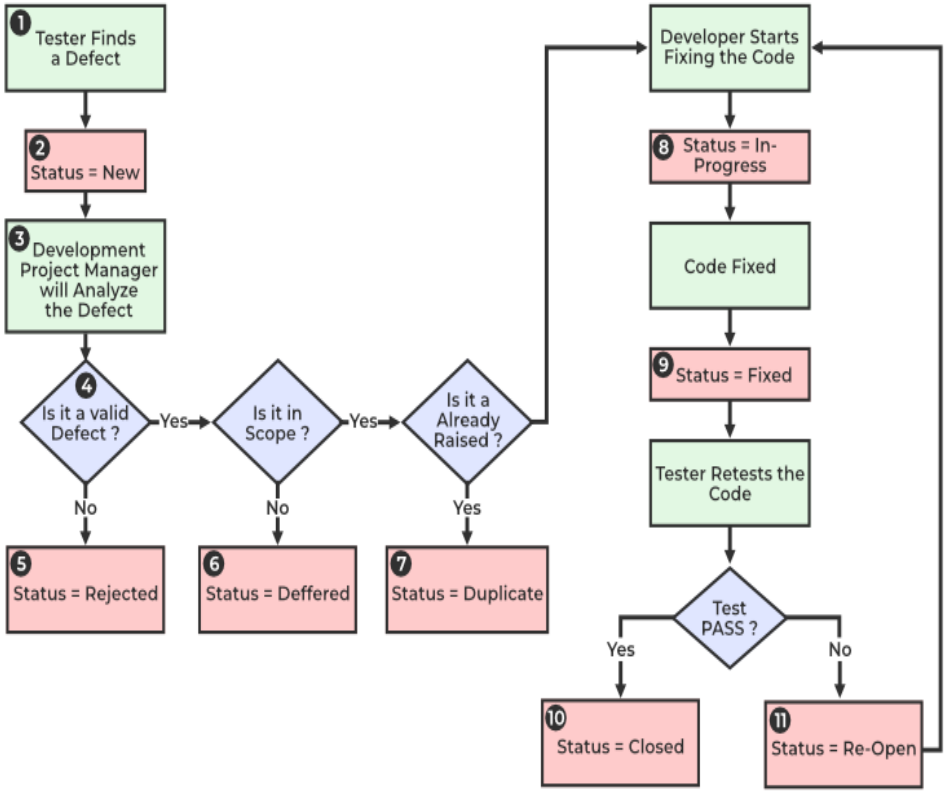


Defect/ Bug Lifecycle

Consider the flow chart below to understand the defect lifecycle.

1. The tester finds a defect.
2. The defect status is changed to **New**.
3. The development manager will then analyze the defect.
4. The manager determines if the defect is valid or not.
5. If the defect is not valid then the development manager assigns the status **Rejected** to defect.
6. If the defect is valid then it is checked whether the defect is in scope or not. If no then the defect status is changed to **Deferred**.

7. If the defect is in scope then the manager checks whether a similar defect was raised earlier. If yes then the defect is assigned a status **duplicate.**
8. If the defect is not already raised then the manager starts fixing the code and the defect is assigned the status **In-Progress**.
9. Once the defect is fixed the status is changed to **fixed.**
10. The tester retests the code if the test is passed then the defect status is changed to **closed.**
11. If the test fails again then the defect is assigned status **reopened** and assigned to the developer.



32. What is Difference between SDLC and STLC?

Difference between SDLC and STLC:

SDLC	STLC

SDLC is mainly related to software development.	STLC is mainly related to software testing.
Besides development other phases like testing is also included.	It focuses only on testing the software.
SDLC involves total six phases or steps.	STLC involves only five phases or steps.
In SDLC, more number of members (developers) are required for the whole process.	In STLC, less number of members (testers) are needed.
In SDLC, development team makes the plans and designs based on the requirements.	In STLC, testing team(Test Lead or Test Architect) makes the plans and designs.
Goal of SDLC is to complete successful development of software.	Goal of STLC is to complete successful testing of software.
It helps in developing good quality software.	It helps in making the software defects free.
SDLC phases are completed before the STLC phases.	STLC phases are performed after SDLC phases.
Post deployment support , enhancement , and update are to be included if necessary.	Regression tests are run by QA team to check deployed maintenance code and maintains test cases and automated scripts.
Creation of reusable software systems is the end result of SDLC.	A tested software system is the end result of STLC.

33. What is Difference between Test Case and Test Script?

Difference between Test Case and Test Script:

Test Case	Test Script
Test Case is a step by step procedure to test any functionality of the software application/product.	Test Script is set of instructions or a short program to test any functionality of software application/product.
Test Case is a manual approach of software testing.	Test Script is an automatic approach of software testing.
It is a set up that is used by the tester to test any specific function of the software product.	It is a program developed by the tester, intended to test any specific function of the software product.
Point by point test case configuration encourages tester to test viably.	Automatic testing approach is beneficial for constant execution.
Test Cases are written manually.	Test Scripting is done by scripting format.
Test case is developed in form of templates.	Test script is developed in form of scripting.
If the tester does not have a good understanding of how the program is used or about the recent risks to the program, then it will be difficult to use the test cases properly.	Active software projects frequently change. So testers have to make a continuous effort to update the scripts

	to match the changes of the new product.
Test Case is used in manual testing environment.	Test Script is used in automatic testing environment.
Test Cases are classified as delegated, positive, reusable, negative and UI test cases.	Test Script are characterized as manual test script and automation test scripts.
Test Case comprises of many test qualities like conditions, test data, environment, test suite id, name and others.	In Test Script different commands can be used to develop scripts which is used as a part of performing repeatable and regression testing.
Requires more resources and time.	Requires less time for testing scripts.

34. What is Difference between Test Plan and Test Scenarios ?

The below table summarizes the difference between the test case and the test scenario:

Parameters	Test Case	Test Scenarios
Definition	Test cases contain definite test steps, data, and expected outcomes for testing all the features of the software under test.	A test scenario is a high-level document that describes end-to-end functionality to be tested.
Testing focus	They are focused on what to test and how to test.	Test scenarios are focused on what to test.

Action Level	These are low-level actions.	These are high-level actions.
Purpose	The aim that is main regarding the test case is to verify the test situation by applying steps.	Writing the test scenario's primary objective is an address end to get rid of the functionality of a software program.
Time requirement	It takes more time in comparison to trying circumstances.	It will take less time as compared to test cases.
Maintenance	The test cases are hard to maintain.	Test scenarios are really easy to maintain due to their high-level design.
Way of testing	Test case helps in exhaustive testing of the application.	Test scenario helps in agile way of testing of the application.
Resource requirement	to write the test, we need extra resources to generate and do test situations.	Fewer resources are sufficient to write test scenarios.
Derived from documents	Test cases are derived from test scenarios.	Test scenarios are usually derived from documents like SRS, BRS, etc.
Ambiguity	There is no ambiguity in test cases as they define test steps, prerequisites, and expected outcomes.	Test scenarios can be ambiguous as these are one-liners.

35. What is Difference between Authorization and Authentication in web testing?

Difference between Data Privacy and Data Protection :

S.No.	Data Privacy	Data Protection
01.	Data Privacy refers maintaining secrecy or keeping control on data access.	Data Protection is the process of protecting data from external risks such corruption, loss etc.
02.	It is all about authorized access means it defines who has authorized access to data.	It is all about unauthorized access means if anyone has not access to data then it keeps the data safe from that unauthorized access.
03.	Data Privacy is a legal process/situation which helps in establishing standards and norms about accessibility.	Data Protection is a technical control system which keeps data protected from technical issues.
04.	Data Privacy is the regulations or policies.	Data protection is the procedures and mechanism.
05.	It can be said as a security from sales means holding the data from shared and sold.	It can be said as s security from hacks means keeping the information away from hackers.
06.	Data Privacy controls are mainly exits at the end user level. The users knows	Data Protection is mainly controlled by the organization or company end. They tech all the required measures to

	which data is shared with whom and which data they can access.	protect their data from being exposed to illegal activities.
07.	Data privacy teams are made of experts with law making, policies and some engineering experts.	Data protection teams are made of experts from technical background, security background etc.

36. What is Methodologies in Agile Development Model?

Agile Testing is a type of software testing that follows the principles of agile software development to test the software application. All members of the project team along with the special experts and testers are involved in agile testing.

Some of the agile testing methodologies are:

1. **Test-Driven Development (TDD):** TDD is the software development process relying on creating unit test cases before developing the actual code of the software. It is an iterative approach that combines 3 operations, programming, creation of unit tests, and refactoring.
2. **Behavior Driven Development (BDD):** BDD is agile software testing that aims to document and develop the application around the user behavior a user expects to experience when interacting with the application. It encourages collaboration among the developer, quality experts, and customer representatives.
3. **Exploratory Testing:** In exploratory testing, the tester has the freedom to explore the code and create effective and efficient software. It helps to discover the unknown risks and explore each aspect of the software functionality.
4. **Acceptance Test-Driven Development (ATDD):** ATDD is a collaborative process where customer representatives, developers, and testers come together to discuss the requirements, and potential pitfalls and thus reduce the chance of errors before coding begins.

5. **Extreme Programming (XP):** Extreme programming is a customer-oriented methodology that helps to deliver a good quality product that meets customer expectations and requirements.
6. **Session-Based Testing:** It is a structured and time-based approach that involves the progress of exploratory testing in multiple sessions. This involves uninterrupted testing sessions that are time-boxed with a duration varying from 45 to 90 minutes. During the session, the tester creates a document called a charter document that includes various information about their testing.
7. **Dynamic Software Development Method (DSDM):** DSDM is an agile project delivery framework that provides a framework for building and maintaining systems. It can be used by users, developers, and testers.
8. **Crystal Methodologies:** This methodology focuses on people and their interactions when working on the project instead of processes and tools. The suitability of the crystal method depends on three dimensions, team size, criticality, and priority of the project.

37. What is Difference between Authorization and Authentication in web testing?

Parameters	JIRA	Bugzilla
Developed	Jira was developed by Atlassian in 2002	Bugzilla was developed by Mozilla in 1998.
License	One needs a commercial license to use this software.	It is an open-source tool.

Server-Side Architecture	The server side architecture of Jira is built on Oracle, MySQL, Perl and PostgreSQL.	The server side architecture of Bugzilla is built on J2EE, Tomcat, Lucene, PostgreSQL, Oracle and MySQL.
Server Load	Server Load in Jira is high as compared to Bugzilla.	Server Load is low in Bugzilla when compared with Jira.
Field Types	There are many custom field types available in Jira and even more can be added with the help of plugins.	There are many custom field types available in Jira and even more can be added with the help of plugins.
Link types	It has configurable link types available.	It has only one link type.
User Interface	It's user interface is a lot better than Bugzilla.	It's user interface has not changed much over the years.
Drag and Drop	The feature "Drag and Drop issue prioritization" is available in Jira.	This feature is not available in Bugzilla.
Customization of Dashboard	The feature "Dashboard with customized gadgets" is available to customize the gadgets according to one's own needs.	This feature is not available in Bugzilla.

Real time tracking	Real-time release tracking feature is available in Jira.	This feature is not available in Bugzilla.
File Attachments	At times, only single file is attached.	At times, only single file is attached.
Kanban Support	Kanban projects are supported in Jira.	Kanban projects are not supported in Bugzilla.
Search Power	It has very powerful search features because of the presence of Jira query language.	It also provides advanced search features but still is less powerful than Jira.

38. Different objects' Scenarios :

Pen

- Verify that the length and the diameter of the pen are as per the specifications.
- Verify the outer body material of the pen. Check if it is metallic, plastic, or any other material specified in the requirement specifications.
- Check the color of the outer body of the pen. It should be as per the specifications.
- Verify that the brand name and/or logo of the company creating the pen should be clearly visible.
- Verify that any information displayed on the pen should be legible and clearly visible.

Functional test cases are the test cases that involve testing the different functional requirements of the object under test.

- Verify the type of pen, whether it is a ballpoint pen, ink pen, or gel pen.
- Verify that the user is able to write clearly over different types of papers.
- Check the weight of the pen. It should be as per the specifications. In case not mentioned in the specifications, the weight should not be too heavy to impact its smooth operation.
- Verify if the pen is with a cap or without a cap.
- Verify the color of the ink on the pen.
- Check the odor of the pen's ink on writing over a surface.
- Verify the surfaces over which the pen is able to write smoothly apart from paper e.g. cardboard, rubber surface, etc.
- Verify that the text written by the pen should have consistent ink flow without leaving any blob.
- Check that the pen's ink should not leak in case it is tilted upside down.
- Verify if the pen's ink should not leak at higher altitudes.
- Verify if the text written by the pen is erasable or not.
- Check the functioning of the pen by applying normal pressure during writing.
- Verify the strength of the pen's outer body. It should not be easily breakable.
- Verify that text written by pen should not get faded before a certain time as mentioned in the specification.
- Check if the text written by the pen is waterproof or not.
- Verify that the user is able to write normally by tilting the pen at a certain angle instead of keeping it straight while writing.
- Check the grip of the pen, and whether it provides adequate friction for the user to comfortably grip the pen.
- Verify if the pen can support multiple refills or not.
- In the case of an ink pen, verify that the user is able to refill the pen with all the supported ink types.
- For ink pens, verify that the mechanism to refill the pen is easy to operate.

- In the case of a ballpoint pen, verify the size of the tip.
- In the case of a ball and gel pen, verify that the user can change the refill of the pen easily.

  The negative test cases include test cases that check the robustness and the behavior of the application when subjected to unexpected conditions.
- Verify the functioning of a pen at extreme temperatures – much higher and lower than room temperature.
- Verify the functioning of a pen at extreme altitude.
- Check the functioning of a pen at zero gravity.
- Verify the functioning of the pen by applying extreme pressure.
- Verify the effect of oil and other liquids on the text written with a pen.
- Check if the user is able to write with a pen when used against gravity i.e. upside down.
- Verify the functioning of a pen when a user tries to write on unsupported surfaces like glass, plastic, wood, etc.
- Verify if the pen works normally or not when used after immersing in water or any other liquid for some period of time.

- Performance test cases include the test cases that help in quantifying or validating the performance of an application under different conditions.
- Check how fast the user can write with the pen over supported surfaces.
- Verify the performance or the functioning of a pen when used continuously without stopping (Endurance Testing).
- Verify the number of characters a user can write with a single refill in the case of ballpoint & gel pens and with full ink, in the case of ink or fountain pens.

Door

- Verify if the door is single door or bi-folded door.
- Check if the door opens inwards or outwards.
- Verify that the dimension of the doors are as per the specifications.

- Verify that the material used in the door body and its parts is as per the specifications.
- Verify that color of the door is as specified.
- Verify if the door is sliding door or rotating door.
- Check the position, quality and strength of hinges.
- Check the type of locks in the door.
- Check the number of locks in the door interior side or exterior side.
- Verify if the door is having peek-hole or not.
- Verify if the door is having stopper or not.
- Verify if the door closes automatically or not – spring mechanism.
- Verify if the door makes noise when opened or closed.
- Check the door condition when used extensively with water.
- Check the door condition in different climatic conditions- temperature, humidity etc.
- Check the amount of force- pull or push required to open or close the door.

Microwave Owen

- Verify that the dimensions of the oven are as per the specification provided.
- Verify that the oven's material is optimal for its use as an oven and as per the specification.
- Verify that the oven heats the food at the desired temperature properly.
- Verify that the oven heats food at the desired temperature within a specified time duration.
- Verify the ovens functioning with the maximum attainable temperature.
- Verify the ovens functioning with minimum attainable temperature.
- Verify that the oven's plate rotation speed is optimal and not too high to spill the food kept over it.
- Verify that the oven's door gets closed properly.
- Verify that the oven's door opens smoothly.

- Verify the battery requirement of the microwave oven and check that it function's smoothly at that power.
- Verify that the text written over the oven's body is clearly readable.
- Verify that the digital display is clearly visible and functions correctly.
- Verify that the temperature regulator is smooth to operate.
- Verify that the temperature regulator works correctly.
- Check the maximum capacity of the oven and test its functioning with that volume of food.
- Check the oven's functionality with different kinds of food – solid, and liquid.
- Check the oven's functionality with different food at different temperatures.
- Verify the oven's functionality with different kinds of container material.
- Verify that the power cord of the oven is long enough.
- Verify that the usage instruction or user manuals have clear instructions.

Coffee wending machine

- UI scenario – Verify that the dimension of the coffee machine is as per the specification.
- Verify that outer body, as well as inner part's material, is as per the specification.
- Verify that the machine's body color as well brand is correctly visible and as per specification.
- Verify the input mechanism for coffee ingredients-milk, water, coffee beans/powder, etc.
- Verify that the quantity of hot water, milk, coffee powder per serving is correct.
- Verify the power/voltage requirements of the machine.
- Verify the effect of suddenly switching off the machine or cutting the power. The machine should stop in that situation and in power resumption, the remaining coffee should not get come out of the nozzle.

- Verify that coffee should not leak when not in operation.
- Verify the amount of coffee served in single-serving is as per specification.
- Verify that the digital display displays correct information.
- Check if the machine can be switched on and off using the power buttons.
- Check for the indicator lights when the machine is switched on-off.
- Verify that the functioning of all the buttons work properly when pressed.
- Verify that each button has an image/text with it, indicating the task it performs.
- Verify that complete quantity of coffee should get poured in a single operation, no residual coffee should be present in the nozzle.
- Verify the mechanism to clean the system work correctly- foamer.
- Verify that the coffee served has the same and correct temperature each time it is served by the machine.
- Verify that system should display an error when it runs out of ingredients.
- Verify that pressing the coffee button multiple times leads to multiple serving of coffee.
- Verify that there is the passage for residual/extra coffee in the machine.
- Verify that machine should work correctly in different climatic, moistures and temperature conditions.
- Verify that machine should not make too much sound when in operation.
- Performance test – Check the amount of time the machine takes to serve a single serving of coffee.
- Performance test – Check the performance of the machine when used continuously until the ingredients run out of the requirements.
- Negative Test – Check the functioning of the coffee machine when two/multiple buttons are pressed simultaneously.
- Negative Test – Check the functioning of coffee machine with a lesser or higher voltage than required.
- Negative Test – Check the functioning of the coffee machine if the ingredient container’s capacity is exceeded.

Chair

- Verify that the chair is stable enough to take an average human load.
- Check the material used in making the chair-wood, plastic etc.
- Check if the chair's leg are level to the floor.
- Check the usability of the chair as an office chair, normal household chair.
- Check if there is back support in the chair.
- Check if there is support for hands in the chair.
- Verify the paint's type and color.
- Verify if the chair's material is brittle or not.
- Check if cushion is provided with chair or not.
- Check the condition when washed with water or effect of water on chair.
- Verify that the dimension of chair is as per the specifications.
- Verify that the weight of the chair is as per the specifications.
- Check the height of the chair's seat from floor.

Gmail (Inbox Mail)

- Verify that a newly received email is displayed as highlighted in the Inbox section.
- Verify that a newly received email has correctly displayed sender email Id or name, mail subject and mail body(trimmed to a single line).
- Verify that on clicking the newly received email, the user is navigated to email content.
- Verify that the email contents are correctly displayed with the desired source formatting.
- Verify that any attachments are attached to the email and are downloadable.
- Verify that the attachments are scanned for viruses before download.
- Verify that all the emails marked as read are not highlighted.
- Verify that all the emails read as well as unread have a mail read time appended at the end on the email list displayed in the inbox section.

- Verify that count of unread emails is displayed alongside ‘Inbox’ text in the left sidebar of Gmail.
- Verify that unread email count increases by one on receiving a new email.
- Verify that unread email count decreases by one on reading an email (marking an email as read).
- Verify that email recipients in cc are visible to all users.
- Verify that email recipients in bcc are not visible to the user.
- Verify that all received emails get piled up in the ‘Inbox’ section and get deleted in cyclic fashion based on the size availability.
- Verify that email can be received from non-Gmail email Ids like – yahoo, Hotmail etc.

Wristwatch

- Verify the type of watch – analog or digital.
- In the case of an analog watch, check the correctness time displayed by the second, minute, and hour hand of the watch.
- In the case of a digital watch, check the digital display for hours, minutes, and seconds is correctly displayed.
- Verify the material of the watch and its strap.
- Check if the shape of the dial is as per specification.
- Verify the dimension of the watch is as per the specification.
- Verify the weight of the watch.
- Check if the watch is waterproof or not.
- Verify that the numbers in the dial are clearly visible or not.
- Check if the watch is having a date and day display or not.
- Verify the color of the text displayed in the watch – time, day, date, and other information.
- Verify that clock’s time can be corrected using the key in case of an analog clock and buttons in case of a digital clock.
- Check if the second hand of the watch makes ticking sound or not.
- Verify if the brand of the watch and check if its visible in the dial.

- Check if the clock is having stopwatch, timers, and alarm functionality or not.
- In the case of a digital watch, verify the format of the watch 12 hours or 24 hours.
- Verify if the watch comes with any guarantee or warranty.
- Verify if the dial has glass covering or plastic, check if the material is breakable or not.
- Verify if the dial's glass/plastic is resistant to minor scratches or not.
- Check the battery requirement of the watch.

Lift (Elevator)

- Verify the dimensions of the lift.
- Verify the type of door of the lift is as per the specification.
- Verify the type of metal used in the lift interior and exterior.
- Verify the capacity of the lift in terms of the total weight.
- Verify the buttons in the lift to close and open the door and numbers as per the number of floors.
- Verify that the lift moves to the particular floor as the button of the floor is clicked.
- Verify that the lift stops when the up/down buttons on a particular floor are pressed.
- Verify if there is an emergency button to contact officials in case of any mishap.
- Verify the performance of the floor – the time taken to go to a floor.
- Verify that in case of power failure, the lift doesn't free-fall and gets halted on the particular floor.
- Verify lifts working in case the button to open the door is pressed before reaching the destination floor.
- Verify that in case the door is about to close and an object is placed between the doors if the doors sense the object and again open or not.
- Verify the time duration for which the door remains open by default.

- Verify if the lift interior is having proper air ventilation.
- Verify lighting in the lift.
- Verify that at no point the lift door should open while in motion.
- Verify that in case of power loss, there should be a backup mechanism to safely get into a floor or a backup power supply.
- Verify that in case the multiple floor number button is clicked, the lift should stop on each floor.
- Verify that in case of capacity limit is reached users are prompted with a warning alert- audio/visual.
- Verify that inside lift users are prompted with the current floor and direction information the lift is moving towards- audio/visual prompt.

What's up Chat

- Verify that the user can set a chat wallpaper.
- Verify that the user sets privacy settings like turning on/off last seen, online status, read receipts, etc.
- Verify that the user can update notification settings like – notification sound, on/off, and show preview for both group and individual chats.
- Verify that the user can take the complete chat backup of his chats.
- Verify that the user can update the phone number that is used by the WhatsApp application.
- Verify that the user can disable/delete his Whatsapp account.
- Verify that the user can check data usage by images, audio, video, and documents in WhatsApp chats.

What's up Group

- Verify that on downloading the Whatsapp application, users can register using a new mobile number.
- Verify that for a new mobile number user will get a verification code on his mobile and filling in the same verifies the new user account.

- Check the maximum number of incorrect attempts allowed while filling out the verification code.
- Verify that registering an existing mobile number for new user account registration is not allowed.
- Verify that on successful registration all the contacts in the user's contact directory get imported to the Whatsapp contact list.
- Verify that the user can set DP and status on Whatsapp.
- Verify that the user can update the existing DP and Whatsapp status.
- Verify that the user can send messages to any individual selected from his contact list.
- Verify that 'Chats' window contains all the chat list with DP and name and last message preview of the other person with whom chat was initiated.
- Verify that clicking a chat in the chat list opens a new window containing all the chats received and sent with the other person.
- Verify that the user can check the message delivered and read the time for a message in the 'Message Info' section.
- Verify that the user can share or receive contact with the other person.
- Verify that the user can create a group by adding multiple people from his contact list.
- Verify that the user can send and receive the message in group chats.
- Verify that users can send and receive images, audio, video, and emoticons in the chat with individuals.
- Verify that users can send and receive images, audio, video, and emoticons in group chats.
- Verify that the user can send and receive chats in the secondary languages available.
- Verify that users can delete text, images, audio, and video messages within a chat.
- Verify that users can clear their complete chat history in an individual or group chat.
- Verify that users can archive chats in an individual or group chat.

- Verify that users can block a user to prevent any message from getting received from the blocked contact.
- Verify that the user makes WhatsApp calls to the person in his contact list.
- Verify that the user can receive WhatsApp calls from the person in his contact list.
- Verify that users can mark chats as favorites and access all chats marked as favorites from the ‘Favorites’ section.

What’s up Payment

- Check whether the application has proper rules and mechanisms in place to authenticate users on the platform.
- Check whether the application has strong password policies. (Especially applicable to banking and finance, social media or e-mail apps)
- Check whether the application has a proper mechanism in place to recover forgotten passwords.
- Check whether the deactivated users and users entering old passwords are not allowed to log in.
- Check whether the confidential data of the user is not accessible to other users on the platform.
- Check whether the confidential data such as payment information and passwords are stored and displayed in an encrypted format.
- Check whether the data entered by the user and data stored by the application is correct.
- Check whether the application can fight against brute force attacks, SQL injection attacks.