1. **What is Error, Defect, Bug and failure?**

* A Human can make an Error.
* An **Error** is ‘A Human Action that produces an Incorrect Result’.
* The error can cause a defect.
* A **Defect** is ‘A flaw in a component or system that can cause the component or system to fail to perform its required function ’.
* A **Defects** can be in the Software, System or in a document.
* Defects occur because human beings are fallible
* Also because of:
* Time pressure
* Complex code
* Complex Infrastructure
* Changed technologies
* And /or many system interactions
* A **Defect** may result in a Failure.
* A **Failure** is a ‘**Deviation of the component or system from its expected delivery, service or result’.**
* **Failures**  can be caused by environmental conditions as well
* E.g. radiation ,magnetism, electronic fields
* Pollution can cause faults in firmware or influence the execution of software by changing hardware conditions

**Errors, Defects and Failures**

**Error**

A Human action that produces an incorrect

result

Can manifest as

A Flaw in a component or system that can cause the component or system to fail perform its required function

**Defects**

**Failure**

May result in

Deviation of the component or system from its expected delivery, service or result

* **Errors, Defects and Failures:**
* **“A mistakes in coding is called error , error found by tester is called defect, defect accepted by development team then it is called bug build does not meet the requirements then it is failure ”**
* **Error:** A discrepancy between a computed, observed, or measured value and condition and the true, specified, or theoretically correct value or condition. This can be misunderstanding of the internal state of the software, an oversight in terms of memory management, conclusion about the proper way to calculate a value, etc.
* **Failure:** The inability of a system or component to perform its required functions within specified performance required. See: bug, crash, exception, and fault.
* **Bug:** A fault in program which causes the program to perform it’s an unintended or unanticipated manner. See: anomaly, defect, error, exception, and fault. Bug is Terminology of Tester.
* **Fault:** An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. See: bug, defect, error, exception.
* **Defect:** Commonly refers to several troubles with the software products, with its external behavior or with its features.

1. **What is a 7key principle? Explain in detail?**

General Testing principles:

1. Testing shows the presence of defects
2. Exhaustive Testing is Impossible!
3. Early testing
4. Defect Clustering
5. The pesticide Paradox
6. Testing is context Dependent
7. Absence of Error fallacy
8. **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 cannot 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

1. **Exhaustive Testing is impossible!**

* 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.

1. **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](https://www.geeksforgeeks.org/requirements-gathering-introduction-processes-benefits-and-tools) phase.

.

1. **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.
* They are clustered.
* In other words, most defects found during testing are usually confined to a small number of modules.

1. **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 Error 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.

**Testing shows o presence of defects**

**Exhaustive Testing is Impossible!**

**Principles of software Testing**

**Absence of error fallacy**

**Testing is context-Dependent**

**Pesticide Paradox**

**Defect Clustering**

**3. Difference between QA vs. QC vs. Tester?**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N** | **Quality Assurance** | **Quality Control** | **Testing** |
| 1 | Activities which ensures the implementation of processes,  Procedures and standards in context to verification of developed software and intended requirements. | Activities which ensures the verification of developed software with respect to documented (or not in some cases) requirements. | Activities which ensures the identification of bugs /error/defects in the software. |
| 2 | Focuses on processes and procedures rather than conducting actual testing on the system. | Focuses on actual testing by executing Software with intend to identify bug /defect through implementation of procedures and process. | Focuses on actual testing. |
| 3 | Process oriented activities. | Product oriented activities. | Product oriented activities. |
| 4 | Preventive activities. | It is a corrective process. | It is a preventive process. |
| 5 | It is a subset of Software Test Life Cycle(STLC) | QC can be considered as the subset of quality Assurance. | Testing is the subset of Quality control. |

**4. Difference between Verification and Validation?**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Verification** | **Validation** |
| **Definition** | The process of evaluating the work products (not the actual final product of a development) phase to determine whether they meet the specified requirements for that phase. | The process of evaluating the software during or at the end of the development process to determine whether it satisfies specified business requirements. |
| **Objectives** | To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements. | The ensure that the product actually meet the user’s needs, and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfils its intended use when placed in its intended environment. |
| **Questions** | Are we building the product right? | Are we building the product right? |
| **Evaluation**  **Items** | Plans ,Requirements, specs, Design Specs, Code, Test cases | The actual product /software. |
| **Activities** | * Reviews * Walkthroughs * Inspections | * Testing |

**5. Explain the difference between Functional Testing and non-functional Testing?**

|  |  |
| --- | --- |
| **Functional Testing** | **Non –Functional Testing** |
| Functional testing is performed using the functional specification provided by the client and verifies the system against them. Functional requirements. | Non –functional testing checks the performance, reliability, scalability and other non-functional aspects of the software system. |
| Functional testing is executed first. | Non –functional testing should be performed after functional testing. |
| Manual Testing and Automation tools can be used for functional testing | Using tools will be effective for this testing. |
| Business requirements are the inputs to functional testing. | Performance parameters :speed, scalability inputs for this testing |
| Functional testing describes what the product does | Nonfunctional testing describes how good the product works. |
| Easy to do manual testing | Tough to do manual testing. |
| Types of Functional Testing are   * Unit Testing * Smoke Testing * Sanity Testing * Integration Testing * Whit box Testing * Black box Testing * User Acceptance Testing * Regression Testing | Types of Nonfunctional Testing are : 1.Performance Testing  2.Load Testing  3.Volume Testing  4.Stress Testing  5.Security Testing  6. Installation Testing  7.Penetration Testing  8.Compatibility Testing  9.Migration Testing |

1. **What is white box testing and list the types of white box testing?**

* White box testing is a [software testing technique](https://www.geeksforgeeks.org/software-testing-techniques/)that involves testing the internal structure and workings of a [software application](https://www.geeksforgeeks.org/what-is-application-software/).
* The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level
* White box testing is also known as [structural testing](https://www.geeksforgeeks.org/structural-software-testing/)or [code-based testing,](https://www.geeksforgeeks.org/what-is-code-driven-testing-in-software-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.
* Before we move in depth of the white box testing do you know that there are many different type of testing used in industry.
* Some automation testing tools are there which automate the most of testing so if you wish to learn the latest industry level tools then you check-out our [manual to automation testing course](https://gfgcdn.com/tu/QW3/)in which you will learn all these concept and tools

## Types of White Box Testing:

White box testing can be done for different purposes. The three main types are : 1.Unit testing

2. Integration Testing

3. Regression Testing

1. **Unit Testing:**

* Checks if each part or function of the application works correctly.
* Ensures the application meets design requirements during development.

1. **Integration Testing:**

* Examines how different parts of the application work together.
* Done after unit testing to make sure components work well both alone and together.

1. **Regression Testing:**

* Verifies that changes or updates don’t break existing functionality.
* Ensures the application still passes all existing tests after updates.
* Objective of White Box Testing:
* White Box Testing serves a crucial role in software testing by allowing testers to inspect and verify the inner workings of a software system, including its code, infrastructure, and integrations.
* **The key objective of White Box Testing includes**:

**1. Thoroughness**: It provides complete coverage, ensuring every part of the software’s internal structure is tested.

2. **Automation**: Test cases can be easily automated, saving time and resources.

**3. Optimization**: It helps in code optimization by identifying hidden errors and redundancies

**4. Introspection**: It provides an in-depth understanding of the software, which can be invaluable for future development and maintenance.

**White box testing techniques:**

One of the main benefits of white box testing is that it allows for testing every part of an application. To achieve complete code coverage, white box testing uses the following techniques:

**Statement Coverage**

**Branch Coverage**

**Path Coverage**

**Decision /Condition Coverage Coverage**

**Condition Coverage**

* 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.
  1. **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.
  1. **Path Coverage:**
* This is like reading a book’s possible combination of chapters.
* In code, it means testing every possible path through the code from start to finish.
  1. **Condition Coverage:**
* This is like checking every possible answer to a question in a book.
* In code, it means testing every possible outcome of logical conditions in the code.
  1. **Decision &Condition Coverage:**
* This is like checking every possible combination of answers to multiple questions in a book.
* In code, it means testing every possible combination of outcomes in logical decisions (like a condition with and logic) in the code.

**7. What is Black Box Testing? What are the different black box testing techniques?**

* Black box testing is software testing techniques where the internal workings or code structure of the system being tested are not known to the tester.
* In other words, the tester focuses solely on the external behavior of the software, without having access to its internal source code. The name “black box” comes from the idea that the internal workings are hidden or “boxed” from the tester’s view.

**Key characteristics of black box testing include:**

* **Independent Testing**: Black box testing is typically performed by testers who are independent of the development team. This ensures a fresh perspective and helps identify issues that developers might overlook.
* **Requirements-Based Testing**: Testers design test cases based on the software’s requirements and specifications, without being concerned about how the code is implemented.
* [**Functional Testing**](https://www.browserstack.com/guide/functional-testing): The main goal of black box testing is to assess the functionality of the software, checking if it meets the expected behavior and delivers the desired outputs for various inputs.
* **No Knowledge of Internal Code**: Testers do not have access to the source code, architecture, or design details of the software. They interact with the system through its user interfaces or APIs.
* **Different Types of Black Box Testing**

Black box testing encompasses several types of testing techniques, each with a specific focus and objective. Some of the main types of black box testing include:

* [**Functional Testing**](https://www.browserstack.com/guide/functional-testing)**:**This type of black box testing verifies that the software’s functions and features work as expected and adhere to the specified requirements. Testers use functional test cases to validate the application’s inputs, outputs, and interactions, without being concerned about the internal code.
* [**Non-Functional Testing**](https://www.browserstack.com/guide/what-is-non-functional-testing): Unlike functional testing, non-functional testing evaluates aspects of the software that are not related to its specific functions. It includes tests for performance, usability, security, scalability, reliability, and other quality attributes.
* [**Regression Testing**](https://www.browserstack.com/guide/regression-testing)**:** Regression testing is performed to ensure that recent changes or updates to the software do not adversely affect existing functionality. Testers use a set of predefined test cases to verify that new features or bug fixes have not introduced new issues.
* **User Interface (UI) Testing:** [UI testing](https://www.browserstack.com/guide/ui-testing-guide) focuses on validating the user interface elements of the software, such as buttons, menus, forms, and layout. The goal is to ensure that the UI is [user-friendly](https://www.browserstack.com/guide/make-website-user-friendly), consistent, and functions correctly.
* [**Usability Testing**](https://www.browserstack.com/guide/what-is-usability-testing)**:** Usability testing assesses the software’s user-friendliness and how easily users can interact with it. Testers evaluate factors like navigation, visual appeal, ease of learning, and overall user experience.
* **Boundary Value Analysis (BVA):** BVA is a technique used to identify defects around the boundaries of input values. Test cases are designed with values at the edges of input ranges to assess how the software handles minimum and maximum limits.
* **Equivalence Partitioning:**In this technique, the input domain is divided into groups of data that are expected to behave similarly. Test cases are then derived from these partitions to minimize redundant testing.
* [**Ad-hoc Testing**](https://www.browserstack.com/guide/adhoc-testing): Ad-hoc testing is an informal and unstructured testing approach where testers explore the software freely, executing test scenarios based on their intuition and experience. It helps identify defects that might be missed by formal test cases.
* [**Compatibility Testing**](https://www.browserstack.com/guide/compatibility-testing): Compatibility testing assesses how well the software performs across different environments, such as various browsers, operating systems, devices, and network configurations.
* **Security Testing:**Security testing aims to identify vulnerabilities and weaknesses in the software’s security measures. Testers simulate attacks and check for potential security breaches.
* **Localization and Internationalization Testing:**These types of testing ensure that the software is adapted to different languages, cultures, and regional settings, and it functions correctly in various international environments.
* **Different Black Box Testing Techniques:**

Black box testing techniques focus on evaluating software from the user’s perspective, without delving into the internal code structure or logic.

**1. Equivalence Partitioning:** Divides the input data into equivalent partitions, with each partition being regarded the same by the program. Testing one representative from each partition is usually enough to cover all potential scenarios.

**Example:** For a form that accepts age input between 18 and 65, equivalence partitions might include:

* Valid partition: 18-65 (e.g., age 25)
* Invalid partition: Below 18 (e.g., age 15)
* Invalid partition: Above 65 (e.g., age 70)

**2. Boundary Value Analysis:** Tests the bounds of input ranges, as errors frequently arise on the edge of input limits.

**Example:** For an input field that accepts values from 1 to 100, boundary values would include:

* Lower boundary: 1
* Just below lower boundary: 0
* Just above upper boundary: 101
* Upper boundary: 100

**3. Decision Table Testing:** A decision table is used to represent and test different combinations of inputs and predicted outcomes. This method is effective for testing systems that involve several conditions and actions.

**Example:** For a loan application system with conditions like credit score (high/low) and income (above/below threshold), a decision table might include:

|  |  |  |
| --- | --- | --- |
| Credit Score | Income | (Loan Approved Y/N) |
| High | Above Threshold | Y |
| High | Below Threshold | Y |
| Low | Above Threshold | N |
| Low | Below Threshold | N |

**4. State Transition Testing:** Tests the system’s behavior in various states and transitions between them. It ensures that the system functions properly when transitioning from one state to another.

**Example:** For a user login system, states might include:

* Logged Out
* Logged In
* Suspended

**Transitions would be:**

* From Logged Out to Logged In (successful login)
* From Logged In to Suspended (suspend account)
* From Suspended to Logged Out (logout from suspended state)

**5. Use Case Testing:** Focuses on validating the functionality of the system based on user interactions described in use cases. It ensures that the system meets the requirements of each use case.

Example: For an online shopping application, a use case might be:

* **Use Case:** Purchase Item
* **Steps:** Select item, add to cart, proceed to checkout, enter payment details, confirm purchase
* **Expected Outcome:** Order confirmation is displayed, and order is recorded

**6. Error Guessing:** Relies on the tester’s experience and intuition to guess where errors might occur based on common mistakes, past experiences, and known problem areas.

**Example:** For a file upload feature, error guessing might include testing with:

* Files of various types (e.g., .exe, .jpg, .pdf)
* Files with very large sizes
* Files with invalid extensions

**7. All-pair Testing Technique:** All-pair testing, also known as pair wise testing, is a combinatorial testing technique that aims to cover all possible pairs of input parameters in a test set.

The purpose is to ensure that every combination of two input parameters is evaluated at least once, which aids in the detection of problems caused by interactions between parameter pairs.

**Consider a web application with three input parameters:**

* **Parameter 1:** Browser Type (Chrome, Firefox)
* **Parameter 2:** Operating System (Windows, macros)
* **Parameter 3:** User Role (Admin, Guest)

With each parameter having two possible values, there are 2 x 2 x 2 = 8 possible combinations if tested exhaustively. However, using all-pair testing, you might only need a subset of combinations to cover all pairs of values.

**Possible Test Cases:**

**Browser:** Chrome, OS: Windows, Role: Admin

**Browser:** Chrome, OS: macros, Role: Guest.

**Browser:** Firefox, OS: Windows, Role: Guest.

**Browser:** Firefox, OS: macros, Role: Admin

**These test cases ensure that each pair of input values is tested, such as:**

Browser Type and Operating System.

Browser Type and User Role.

Operating System and User Role

**8. Cause-Effect Technique:** The Cause-Effect approach, also known as Cause-Effect Graphing, is a black-box testing method that creates test cases based on the relationships between causes (inputs) and effects (outputs).

This technique aids in systematically determining the functional correctness of a system by visualizing and analyzing the logical linkages between various situations and actions.

**Some key concepts here are:**

* **Cause-Effect Graph:** A diagrammatic portrayal of the logical relationships between various input conditions (causes) and their anticipated outputs (effects). It assists in discovering and mapping how different inputs interact to produce diverse outcomes, allowing for more effective test case creation.
* **Cause:** An input condition or factor that determines system behaviour. Examples include user inputs, system settings, and configuration parameters.
* **Effect:** The result of the cause. It is the system’s response to the provided input. Examples include system outputs, status messages, and changes in system behavior.

**Example:**

Consider an online account login system with the following input conditions (causes) and expected outputs (effects):

**Causes:**

* Correct username
* Correct password
* Incorrect username
* Incorrect password
* Account locked

**Effects:**

* **Login Success:** If the username and password are both correct and the account is not locked.
* **Login Failure:** If either the username or password is incorrect, or the account is locked.

**Cause-Effect Graph:**

* Cause 1 + Cause 2 → Effect 1 (Successful Login)
* Cause 3 + Cause 4 → Effect 2 (Login Failure)
* Cause 5 → Effect 3 (Account Locked)

**Derived Test Cases:**

* Correct username + Correct password (Expected: Login Success)
* Correct username + Incorrect password (Expected: Login Failure)
* Incorrect username + correct password (Expected: Login Failure).
* Account locked + Correct username + Correct password (Expected: Account Locked)
* **Example of Black Box Testing:**

A simple black box testing example for login functionality of a web application. In this scenario, we will test the login page without having access to the internal code or implementation details.

**Test Case Name**: Verify successful login with valid credentials.

**Test Steps**:

1. Open the web browser.
2. Enter the URL of the application’s login page.
3. Enter a valid username in the username field.
4. Enter a valid password in the password field.
5. Click on the “Login” button.
6. Wait for the application to process the login request.

**Expected Result**: The user should be successfully logged into the application’s dashboard/homepage.

**Test Case Status**: PASS (if the user is redirected to the dashboard/homepage)

**Test Case Name**: Verify unsuccessful login with invalid credentials.

**Test Steps**:

1. Open the web browser.
2. Enter the URL of the application’s login page.
3. Enter an invalid username (e.g., “invalid user”) in the username field.
4. Enter an invalid password (e.g., “wrong password”) in the password field.
5. Click on the “Login” button.

Wait for the application to process the login request.

**Expected Resul**t: The login attempt should fail, and an appropriate error message (e.g., “Invalid username or password”) should be displayed on the login page.

**Test Case Status**: PASS (if the error message is displayed)

**8. What Is Functional System Testing?**

* Functional testing is a type of software testing that verifies that each function of the software application behaves according to the specific requirements and meets expectations in different conditions.
* The goal of functional testing is to validate the system’s features, capabilities, and interactions with different components.
* It involves testing the software’s input and output, data manipulation, user interactions, and the system’s response to various scenarios and conditions.
* Functional testing is only concerned with validating if a

System works as intended.

* Functional testing checks an application, website, or system to ensure that it is doing exactly what it is meant to.
* Unlike other types of testing that might focus on performance, security, or usability, functional testing is primarily concerned with the correctness of the application’s functionality.

**Example of Functional Testing:** Consider the example of testing a login feature. When you perform functional testing in this context, you verify that users can log in when the credentials are submitted, see the right error messages upon submitting incorrect credentials and reset their password easily.

**Benefits of Functional Testing:**

* Helps in ensuring the software fulfills user needs
* Helps identify functional issues early on
* Enhances application quality and user experience
* Ensures application compatibility across different platforms
* Verifies user workflows and interactions

**Types of Functional Testing:**

**Here are the types of functional testing:**

* 1. **Unit Testing:**
* This is performed by developers who write scripts that test if individual components/units of an application match the requirements.
* This usually involves writing tests that call the methods in each unit and validate them when they return values that match the requirements.
* In unit testing, [code coverage](https://www.browserstack.com/guide/code-coverage-techniques) is mandatory. Ensure that test cases exist to cover the following:
* Line Coverage
* Code path Coverage
* Method Coverage
  1. **Smoke Testing:**

This is done after the release of each build to ensure that software stability is intact and not facing any anomalies.

* 1. **Sanity Testing:**

Usually done after smoke testing, this is run to verify that every major functionality of an application is working perfectly, both by itself and in combination with other elements.

* 1. **Regression Testing:**

This test ensures that changes to the codebase (new code, debugging strategies, etc.) do not disrupt the already existing functions or trigger some instability.

* 1. **Beta/Usability Testing:**

In this stage, actual customers test the product in a production environment. This stage is necessary to gauge how comfortable a customer is with the interface. Their feedback is taken for implementing further improvements to the code.

* 1. **White Box Testing:**

In White box testing, the tester validates the internal functioning of a software system.

* 1. **Integration Testing:**
* If a system requires multiple functional modules to work effectively, integration testing is done to ensure that individual modules work as expected when operating in combination with each other.
* It validates that the end-to-end outcome of the system meets these necessary standards.

**8. Grey Box Testing:**

Grey-box testing Integrates aspects of both black-box and white-box testing, where the internal functions of the applications are partially tested. The functionality and security are tested here by utilizing the system’s architecture.

**9. Exploratory Testing:**

In exploratory testing, testers explore the software to detect issues and evaluate user experience without depending on predefined test cases.

**10. Black box Testing:**

In Black box testing, the software system’s functionality is validated without examining its internal functions. Testers assess the inputs, outputs, and user interactions to ensure the systems fulfill requirements.

**11. Component Testing:**

* Component testing is performed right after unit testing.
* In this approach, test objects are tested independently as a component without combining with other components.

**12. Database Testing:**

Database testing verifies the accuracy and reliability of data within a database by making sure that database aspects like schema, CRUD (Create, Read, Update, Delete) etc. function as expected under different conditions.

**13. Recovery Testing:**

Recovery testing validates the application’s ability to recover from various types of failures.

**14. Static Testing:**

Static testing is a type of functional testing approach that includes the assessment of the software’s code, design, or documentation without actually executing the program.

**9. What is Non-Functional Testing?**

* Non-functional testing is essential for confirming the software’s reliability and functionality.
* Non-functional testing focuses on evaluating the system’s performance, scalability, security, usability, and reliability, rather than its specific functionality.
* It ensures that the system can handle real-world demands and provides a seamless, high-quality user experience.
* **Examples of non-functional testing include:**
* Performance testing
* Scalability testing
* Usability testing
* Security testing
* Localization testing
* Reliability testing
* **Core Objectives of Non-Functional Testing:**
* Below are some core objectives of non-functional testing and their explanation in detail.
* Enhancing Usability and Effectiveness
* Minimizing Production Risks and Costs
* Improving Product Setup and Operations
* Measuring and Analyzing Metrics
* Understanding Technology and Product Behavior

**Here are the objectives in detail:**

* **Enhancing Usability and Effectiveness**: Ensures the product is user-friendly, efficient, maintainable, and portable across various environments.
* **Minimizing Production Risks and Costs**: Reduces risks and costs by addressing non-functional issues early, preventing failures post-deployment.
* **Improving Product Setup and Operations**: Ensures smooth installation, configuration, execution, and effective product management and monitoring.
* **Measuring and Analyzing Metrics**: Collects performance metrics to analyze system behavior and drive optimization and development.
* **Understanding Technology and Product Behavior**: Offers insights into product behavior with current technology, ensuring compatibility and performance.
* **Characteristics Of Non-Functional Testing**
* Evaluating a system’s quality involves assessing various attributes that impact its performance and user experience. Some characteristics of non-functional testing include:
* Non-functional testing should be quantifiable. Therefore adjectives like “good,” “better,” “best,” etc., have no place in this type of testing.
* It is doubtful that exact figures will be known at the beginning of the requirement process.
* It’s crucial to prioritize the requirements.
* Make sure that in software engineering, quality attributes are accurately identified.

**Types of Non-Functional Testing:**

Here are various types of non-functional testing, each designed to assess specific aspects of a system’s performance and quality. Here’s a closer look at these types and their purpose.

#### 1. Performance Testing

* [Performance testing](https://www.browserstack.com/guide/performance-testing) eliminates the causes of the software’s sluggish and constrained performance. The software’s reading speed should be as quick as possible. One must create a well-organized and precise specification about the desired speed for Performance Testing. Otherwise, it won’t be evident if the test is a success or a failure. Example: When 1000 users use an application simultaneously, the load time shouldn’t exceed 5 seconds.

**Tools Used:** Roadrunner, Apache J Meter, Web LOAD.

#### 2. Load Testing

The system’s loading capability is tested during load testing. The system can handle increasing simultaneous users because of its loading capacity. If you want*to run a quick website speed test,*[*check your speed scores*](https://www.browserstack.com/speedlab)*.*

**Tools Used:** Neoload, Load Multiplier.

#### 3. Security Testing

Security testing is used to find the software application’s weaknesses. The testing is carried out by looking into the design of the system and the mindset of an attacker. Finding the parts of the code where an attack is most likely to occur allows for [creating test cases](https://www.browserstack.com/guide/test-cases-for-automated-tests).

**Tools Used:** ImmuniWeb, Vega, Wapiti

#### 4. Portability Testing

The software’s ability to run on many operating systems without experiencing any bugs is tested for portability. The software’s functionality under the same operating system but with varied hardware is also tested in this test.

**Tools Used:** SQLMap.

#### 5. Accountability Testing

Accountability testing helps identify if the system is functioning correctly or not. A function ought to produce the same outcome for which it was designed. The system passes the test if it produces the desired results; else, it fails.

**Tools Used:** Mentimeter.

#### 6. Reliability Testing

This testing assumes that the software system operates without error under the predetermined parameters. A certain amount of time and processes must be used to run the system. The reliability test will also fail if the system fails under certain predetermined circumstances. For example, all the web pages and links should be reliable.

**Tools Used:** Test-retest, Inter-rater.

#### 7. Efficiency Testing

This testing looks at how many resources were used and how many were required to construct a software system.

* **Tools Used:** Web LOAD, Load Ninja.

#### 8. Volume Testing

A [type of software testing](https://www.browserstack.com/guide/types-of-testing) called volume testing involves exposing the software to a significant amount of data. It is additionally known as flood testing. By increasing the data in the database, volume testing is done to evaluate the system’s performance.

**Tools Used:** HammerDB, JdbcSlim

#### 9. Recovery Testing

Recovery testing evaluates how well an application can bounce back from crashes, hardware failures, and other issues of a similar nature. Recovery testing involves intentionally breaking the software in several different ways.

**Tools Used:** Box Backup, Bacula.

#### 10. Responsive Testing:

[Responsive Testing](https://www.browserstack.com/responsive) allows you to evaluate your design on many screen widths for a “real” test of “adaptivity” instead of predefined screen widths. Test for responsiveness by DesignModo You can adjust the screen’s width after entering the website’s URL to watch how your user interface adapts to accommodate it in real time. A smooth experience on various digital devices is the main objective of evaluating responsive websites. We increasingly rely on technology to get things done because of the convenience it has brought to the globe.

**Tools Used:** Responsinator, Screen fly, Google DevTools Device Mode

#### 11. Visual Testing

The solution to such issues is visual testing, sometimes known as [visual UI testing](https://www.browserstack.com/guide/visual-testing-beginners-guide). It checks if every user sees the software user interface (UI) correctly. Visual tests verify that each element on a web page has the proper shape, size, and placement. [Visual testing](https://www.browserstack.com/guide/visual-testing-beginners-guide) assesses an application’s visible output and compares it to the outcomes anticipated by design. In other words, it assists in identifying “[visual bugs](https://www.browserstack.com/guide/what-are-visual-bugs)“, which are separate from strictly functional bugs, in the way a page or screen appears.

**Tools Used:** Percy, PhantomCSS, FBSnapshotTestCase, Gemini, Needle (Uses Python).

**Non-Functional Testing Parameters:**

Listed below are the non-functional testing parameters:

1. **Security:** This parameter specifies how a system is protected from planned and unplanned intrusions from both internal and external sources. This is examined through security testing.
2. **Reliability:** The consistency with which a software system consistently completes the required tasks without error. Reliability Testing is used to test this.
3. **Efficiency:** The capacity, quantity, and response time a software system can manage.
4. **Usability**: The simplicity with which a user can engage with a system, learn how to use it, and prepare inputs and outputs. Usability testing verifies this.
5. **Availability**: The parameter establishes the user’s dependence on the system during operation. Testing for stability verifies this.
6. **Scalability:** The word describes how much a software program can raise its processing power to handle a rise in demand. This is examined through scalability testing.

* **Advantages of Non-Functional Testing:**

. By addressing critical aspects of system performance and usability, non-functional testing offers several key benefits that contribute to overall software quality.

* **Enhanced Security**: Ensures systems are protected against online threats, safeguarding data and user trust.
* **Improved Load Handling**: Guarantees the system’s ability to handle concurrent users without performance issues.
* **Increased Efficiency**: Optimizes the system to perform tasks quickly and effectively under various conditions.
* **Reusable Test Cases**: Test cases remain unchanged, eliminating the need to rewrite them repeatedly.
* **Reduced Time Commitment**: Requires less time compared to other testing procedures, streamlining the testing process.

**10. What is Exploratory System?**

* Exploratory Testing is an unscripted approach to software testing where testers actively explore the application to discover bugs, learn its behavior, and design tests in real time without predefined test cases.
* In exploratory testing, testers do not work based on previously created test cases.
* They check a system without a plan to discover bugs that users may face when navigating a website or app without a specific aim or direction.

**Benefits of the exploratory System?**

* Uncovers **hi**dden bugs and edge cases.
* Adapts to changing requirements.
* Encourages creativity and critical thinking.
* Provides quick feedback.
* Requires minimal preparation.
* Enhances overall test coverage.
* Complements automated and scripted testing.

**Why use exploratory system?**

* Exploratory Here are some of the top reasons why you must implement testing.
* Reasons Why You Must Use Exploratory Testing. Identifying Early Bugs:
  + Gathering User Feedback
  + Evaluating from a User’s Viewpoint
  + Revealing Hidden Issues. Flexible Testing Methods
* Encouraging Creativity
* Ideal for Agile Environments
  + Quick Understanding

## When should you use Exploratory Testing?

* **When need to learn quickly about the application:** Exploratory testing is beneficial for the scenarios when a new tester enters the team and needs to learn quickly about the application and provide rapid feedback.
* **Review from a user perspective:** It comes in handy when there is a need to review products from a user perspective.
* **Early iteration required:** Exploratory testing is helpful in scenarios when an early iteration is required as the teams don’t have much time to structure the test cases.
* **Testing mission-critical applications:**Exploratory testing ensures that the tester doesn’t miss the edge cases that can lead to critical quality failures.
* **Aid unit test:** Exploratory testing can be used to aid unit tests, document the test cases, and use test cases to test extensively during the later sprints.

## When to say no to exploratory testing:

* Organizations must be able to get the proper balance between exploratory testing and scripted testing.
* Until you reach a proper initial state only exploratory testing will not work and will not cover the expected result for the team.
* Especially when with any type of testing that is regulated the compliance-based scripted testing is beneficial to use at that time.
* In compliance testing, many certain checklists and mandatory to follow the legal reason.
* It’s best to use scripted testing where several laws govern the testing protocol and some standards are needed to match.

## Types of Exploratory Testing:

There are many types of exploratory testing. Few are as follows:

1. **Freestyle:**In freestyle exploratory testing, the application is tested in an ad-hoc way, there is no maximum coverage, and there are no rules to follow for testing. It is done in the following cases:
   1. When there is a need to get friendly with the application.
   2. To check other test engineers’ work.
   3. To perform smoke tests quickly.
2. **Strategy Based: Strategy-based** testing can be performed with the help of multiple testing techniques like decision-table testing, cause-effect graphing, boundary value analysis, equivalence partitioning, and error guessing. It is done by an experienced tester who has known the application for the longest time.
3. **Scenario Based:** Scenario-based exploratory testing is done based on scenarios with the help of multiple scenarios like end-to-end, test scenarios. The scenarios can be provided by the user or can be prepared by the test team.
4. **Collaborative Exploratory Testing:** During exploration, several testers work together, exchanging observations and insights. Working together improves the variety of viewpoints and raises the possibility of finding various kinds of flaws.
5. **Charter-Based Exploratory Testing:**The charter, a written document that describes the goals and scope of the investigation, is followed by testers. The charter could list particular features to test, situations to consider or goals to accomplish.
6. **Timing-Boxed Exploratory Testing:**There is a temporal limit on how long exploration can last. The goal of testers is to find as many serious flaws as they can in the allotted period.

## Advantages of Exploratory Testing:

* **Less preparation required:**It takes no preparation as it is an unscripted testing technique.
* **Finds critical defects:**Exploratory testing involves an investigation process that helps to find critical defects very quickly.
* **Improves productivity:**In exploratory testing, testers use their knowledge, skills, and experience to test the software. It helps to expand the imagination of the testers by executing more test cases, thus enhancing the overall quality of the software.
* **Generation of new ideas:**Exploratory testing encourages creativity and intuition thus the generation of new ideas during test execution.
* **Catch defects missed in test cases:**Exploratory testing helps to uncover bugs that are normally ignored by other testing techniques.

## Disadvantages of Exploratory Testing:

* **Tests cannot be reviewed in advance:** In exploratory testing, Testing is performed randomly so once testing is performed it cannot be reviewed.
* **Dependent on the tester’s knowledge:** In exploratory testing, the testing is dependent on the tester’s knowledge, experience, and skill. Thus, it is limited by the tester’s domain knowledge.
* **Difficult to keep track of tests:**In Exploratory testing, as testing is done in an ad-hoc manner, keeping track of tests performed is difficult.
* **Not possible to repeat test methodology:** Due to the ad-hoc nature of testing in exploratory testing, tests are done randomly and thus it is not suitable for longer execution time, and it is not possible to repeat the same test methodology.

## Exploratory Testing Process:

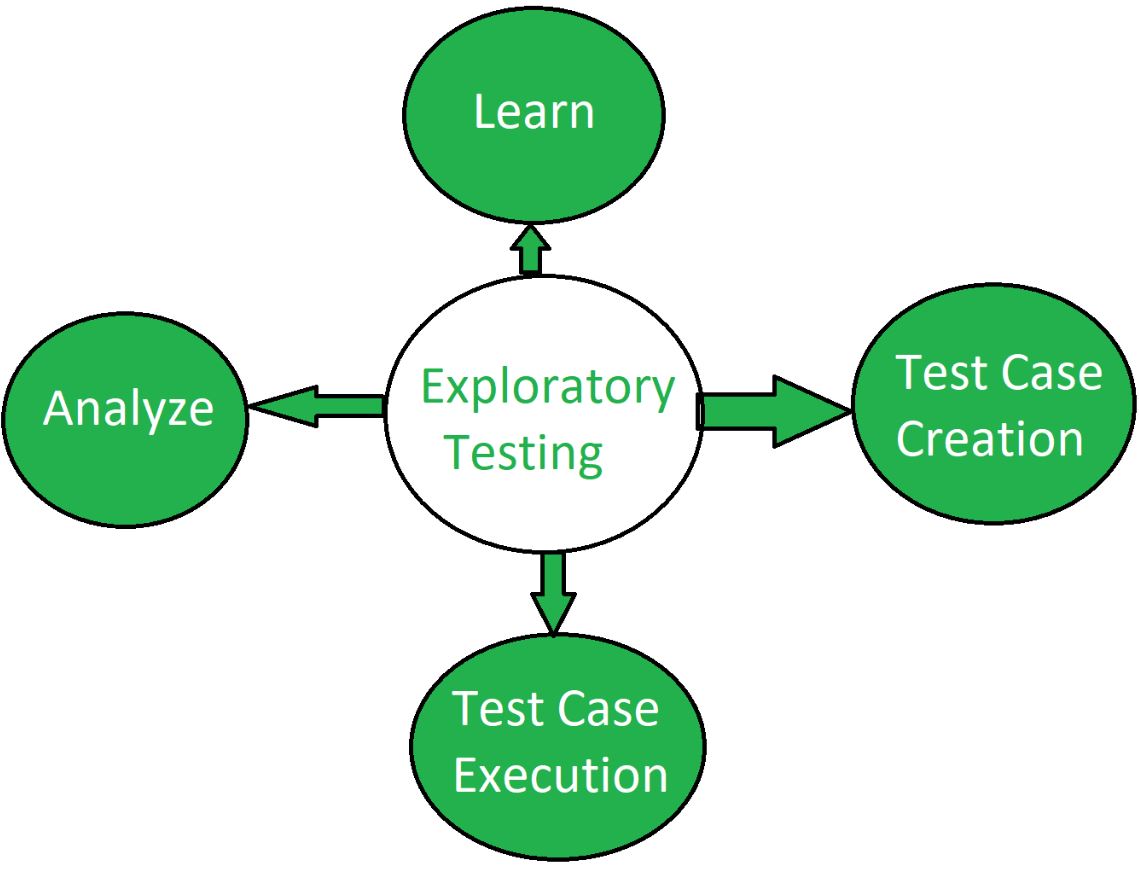
The following 4 steps are involved in the exploratory testing process:

1. **Learn:** This is the first phase of exploratory testing in which the tester learns about the faults or issues that occur in the software. The tester uses his/her knowledge, skill, and experience to observe and find what kind of problem the software is suffering from. This is the initial phase of exploratory testing. It also involves different new learning for the tester.
2. **Test Case Creation:** When the fault is identified i.e. tester comes to know what kind of problem the software is suffering from then the tester creates test cases according to defects to test the software. Test cases are designed by keeping in mind the problems end users can face.
3. **Test Case Execution:** After the creation of test cases according to end user problems, the tester executes the test cases. Execution of test cases is a prominent phase of any testing process. This includes the computational and operational tasks performed by the software to get the desired output.
4. **Analysis:** After the execution of the test cases, the result is analyzed and observed whether the software is working properly or not. If the defects are found then they are fixed and the above three steps are performed again. Hence this whole process goes on in a cycle and software testing is performed.

**Conclusion:**

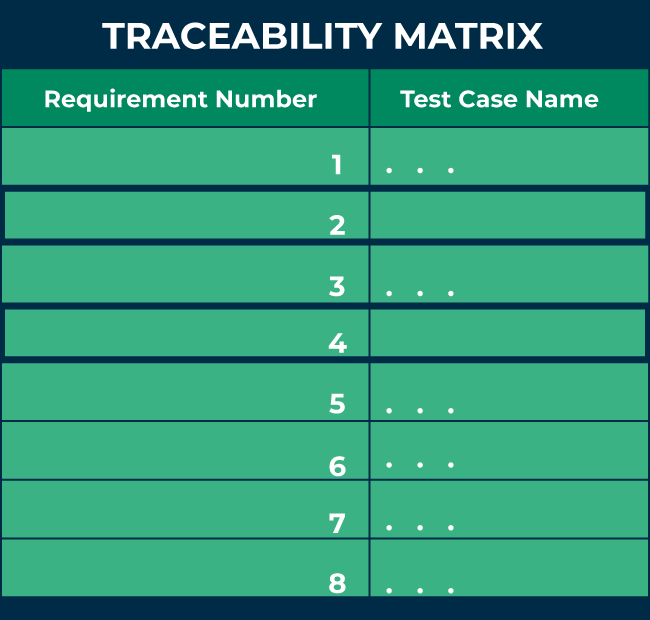
Many advantages come with exploratory testing, such as its adaptability, versatility and capacity to address unexpected scenarios. This method helps testers to swiftly evaluate the functionality of the software and spot possible problems, which makes it especially useful early in a project or when dealing with changing requirements. A software product that incorporates exploratory testing within the testing lifecycle is more robust and dependable.

**Fig: Exploratory System is given below:**

****

11. **What is traceability matr**ix?

* RTM stands for **Requirement Traceability matrix**.
* RTM maps all the requirements with the test cases.
* By using this document one can verify test cases cover all functionality of the application as per the requirements of the customer.
* **Requirements:** Requirements of a particular project from the client

**Traceability:** The ability to trace the tests.

**Matrix:** The data which can be stored in rows and columns form.

The main purpose of the requirement traceability matrix is to verify that the all requirements of clients are covered in the test cases designed by the testers.  
In simple words, one can say it is a pen and pencil approach i.e., to analyze the two data information but here we are using an Excel sheet to verify the data in a requirement traceability matrix.

## Why is ****Requirement Traceability Matrix****(RTM) Important?

* When business analysis people get the requirements from clients, they prepare a document called [SRS (System/Software Requirement Specification)](https://www.geeksforgeeks.org/software-requirement-specification-srs-format/) and these requirements are stored in this document.
* If we are working in the [Agile model](https://www.geeksforgeeks.org/software-engineering-agile-development-models/), we call this document Sprint Backlog, and requirements are present in it in the form of user stories.
* When QA gets the SRS/Sprint backlog document they first try to understand the requirements thoroughly and then start writing test cases and reviewing them with the entire project team.
* But sometimes it may happen that in these test cases, some functionality of requirements is missing, so to avoid it we required a requirement traceability matrix.
* Each [test case](https://www.geeksforgeeks.org/software-testing-test-case/) is traced back to each requirement in the RTM. Therefore, there is less chance of missing any requirement in testing, and 100% test coverage can be achieved.
* RTM helps users discover any change that was made to the requirements as well as the origin of the requirement.
* Using RTM, requirements can be traced to determine a particular group or person that wanted that requirement, and it can be used to prioritize the requirement.
* It helps to keep a check between requirements and other development artifacts like technical and other requirements.
* The Traceability matrix can help the tester identify whether by adding any requirement previous requirements are affected or not.
* RTM helps in evaluating the effect on the QA team to reuse the test case.

## Parameters of ****Requirement Traceability Matrix****(RTM):

* The below figure shows the basic template of RTM. Here the requirement IDs are row-wise and test case IDs are column-wise which means it is a forward traceability matrix.
* From the figure below, it can be seen that: RTM

**The following are the parameters to be included in RTM:**

1. **Requirement ID:** The requirement ID is assigned to every requirement of the project.
2. **Requirement description:** for every requirement a detailed description is given in the SRS (System/Software Requirement Specification) document.
3. **Requirement Type:** understand the type of requirements i.e., banking, telecom, healthcare, traveling, e-commerce, education, etc.
4. **Test cases ID:** the testing team designs test cases. Test cases are also assigned with some ID.

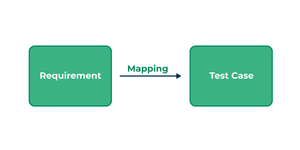
## Types of Traceability Matrix:

**There are 3 types of traceability matrix:**

1. **Forward traceability matrix**
2. **Backward traceability matrix**
3. **Bi-directional traceability matrix**

### ****1. Forward traceability matrix:****

* In the forward traceability matrix, we mapped the requirements with the test cases.
* Here we can verify that all requirements are covered in test cases and no functionality is missing in test cases.
* It helps you to ensure that all the requirements available in the SRS/ Sprint backlog can be traced back to test cases designed by the testers.
* It is used to check whether the project progresses in the right direction.



Forward traceability matrix

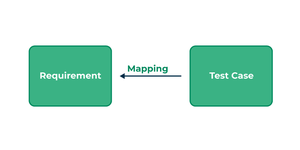
**In forwarding the traceability matrix:**

*Rows = Requirement ID*

*Column = Test case ID*

### ****2. Backward traceability matrix:****

* In the backward traceability matrix, we mapped the test cases with the requirements.
* Here we can verify that no extra test case is added which is not required as per our requirements.
* It helps you to ensure that any test cases that you have designed can be traced back to the requirements or user stories, and you are not extending the scope of the work by just creating additional test cases that cannot be mapped to the requirement.
* The backward traceability matrix is also known as the **reverse traceability matrix**.



Backward traceability matrix

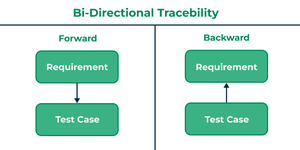
**In the**Excel **backward traceability matrix:**

*Rows = Test cases ID*

*Column = Requirement ID*

### ****3. Bi-directional traceability matrix:****

* A bi-directional traceability matrix is a combination of a forward traceability matrix and a backward traceability matrix.
* Here we verify the requirements and test cases in both ways.



Bi-directional traceability matrix

Bi-directional traceability matrix = Forward traceability matrix + Backward traceability matrix

**Who Needs Requirement Traceability Matrix (RTM)?**

When testers design the test cases they need to check whether test cases cover all functionality of the application as per the requirements of the customer given in the SRS/Sprint backlog.

* To verify that they need a requirement traceability matrix.
* They generally use an Excel sheet or Google spreadsheet for RTM.

## How To Create RTM?

Before creating RTM SRS/Sprint backlog documents and test cases documents are required. Below are the steps to create RTM:

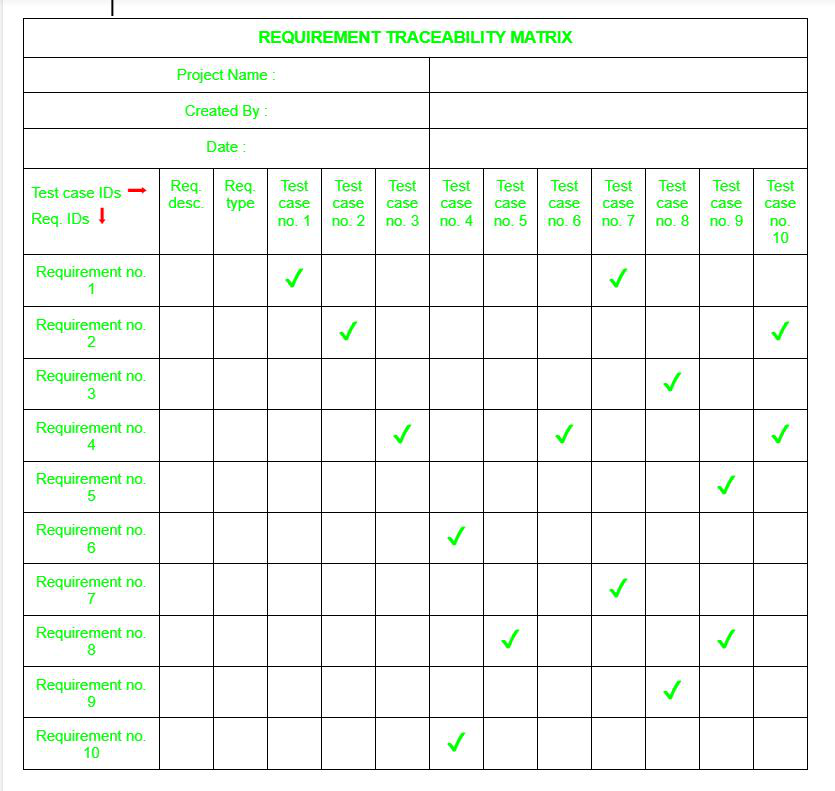
1. For RTM we will use an Excel sheet.
2. Write the name of the project, date, and name of the person who is responsible for RTM.
3. Write all requirement IDs row-wise in the first column of an Excel sheet.
4. Write all the requirement descriptions row-wise in the second column of an Excel sheet.
5. Write all the requirements type row-wise in the third column of an Excel sheet.
6. Write all the test cases with their IDs column-wise in an Excel sheet.
7. After writing all requirements and test cases you have to verify that for every requirement you have prepared the test cases in both positive and negative flow.

## Advantages of RTM:

Below are some benefits of using RTM:

1. **Full test coverage:**RTM confirms the 100% test coverage.
2. **Verify missing functionality:**This document is helpful for the tester to check there is not any functionality missed while testing the application.
3. **Helps to prioritize and track requirements:**It also helps to understand what extra test cases we added that are not part of the requirement.
4. **Helps to track test status:**It is easy to keep track of the overall test status.
5. **Proper consistent documentation:**RTM can help in the effort to provide proper and consistent documentation for the team.
6. **Versioning is easier:**RTM helps to keep track of the required modifications and how they impact every part of the project.

## ****Requirement Traceability Matrix****(RTM) Template:

The below figure shows the basic template of RTM. Here the requirement IDs are row-wise and test case IDs are column-wise which means it is a forward traceability matrix.

From the figure below, it can be seen that:

* For verifying requirement number 1 there are test cases number 1 and 7.
* In requirement number 2 there are test cases number 2 and 10 and similarly, for all other requirements, there are test cases to verify them.

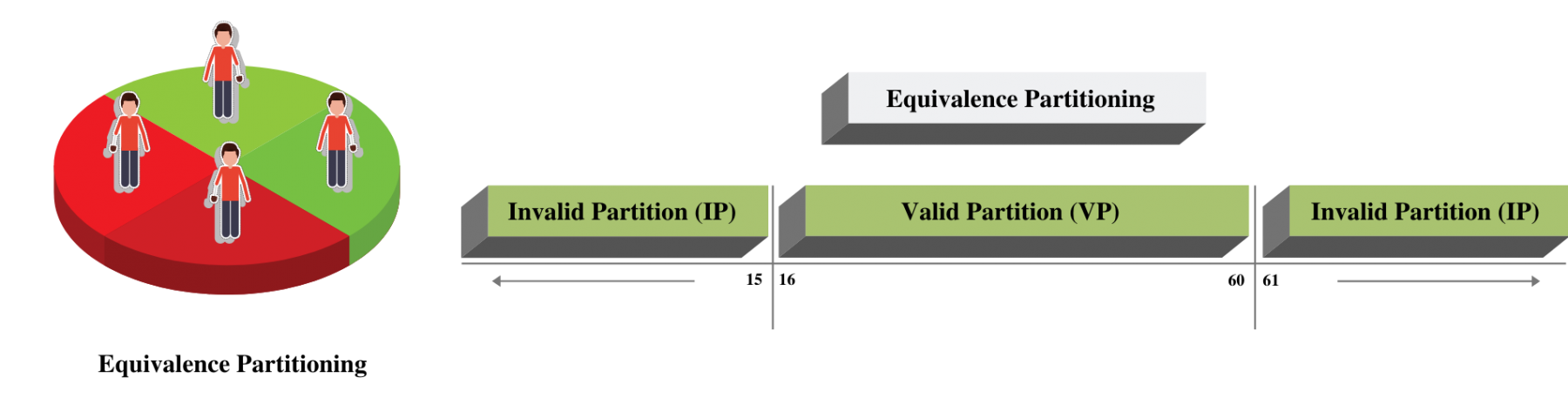
**Q.12 what is Boundary value analysis?**

* The basis of Boundary Value Analysis (BVA) is testing the boundaries at partitions (Remember Equivalence Partitioning!). BVA is an extension of [equivalence partitioning](https://www.toolsqa.com/software-testing/istqb/equivalence-partitioning/).
* However, this is useable only when the partition is ordered, consisting of numeric or sequential data.
* The minimum and maximum values of a partition are its boundary values.
* We have seen that there are high chances of finding the defects at the boundaries of a partition (E.g., A developer using >10 instead of >= 10 for a condition).
* Equivalence partitioning alone was not sufficient to catch such defects. Therefore, a need to define a new technique that can detect anomalies at the boundaries of the partition arose.
* It is how Boundary value analysis came into the picture.
* Boundary value analysis can perform at all [test levels](https://www.toolsqa.com/software-testing/test-levels/), and its primarily used for a range of numbers, dates, and time.

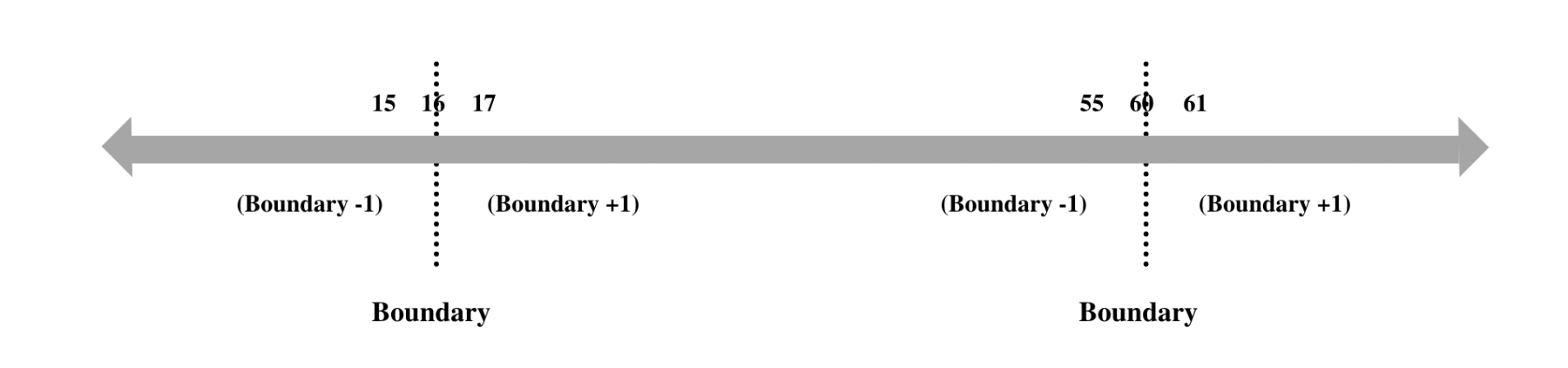
**How to Do Boundary Value Analysis?**

Now that we have got some idea on boundary value analysis let's understand how to derive test conditions using this technique. We will refer to the same example of gym form (Refer to our article on Equivalence Partitioning) where we need to enter Age.

The first step of Boundary value analysis is to create Equivalence Partitioning, which would look like below.



Now concentrate on the Valid Partition, which ranges from 16-60. We have a 3 step approach to identify boundaries:



* Identify Exact Boundary Value of this partition Class - which is 16 and 60.
* Get the Boundary value which is one less than the exact Boundary - which is 15 and 59.
* Get the Boundary Value which is one more than the precise Boundary - which is 17 and 61 . If we combine them all, we will get below combinations for Boundary Value for the Age Criteria.

Valid Boundary Conditions: Age = 16, 17, 59, 60

Invalid Boundary Conditions : Age = 15 61,

It's straightforward to see that valid boundary conditions fall under Valid partition class, and invalid boundary conditions fall under Invalid partition class.

Can you figure out why we have not used 16.1, 15.9, 59.9, and 60.1 as the boundary increment and decrement values? It's a concept that has an insufficient explanation in most of the articles. Therefore, let's take another example to explain this. Assume that you are entering your weight on a website. Based on your weight and height, the site will tell you the Body Mass Index (BMI). You can enter values from 30 to 150 kg in the weight input field. **The weight input field only allows natural numbers i.e., positive integers!**

In this case, if you will create the boundaries using the same method - you will end up with

**Valid Boundary Conditions: Age = 30, 31, 149, 150**

**Invalid Boundary Conditions: Age = 29, 151**

Now consider the same scenario, but the **weight input field allows decimal numbers up to 1 decimal place**. In this case, the boundary conditions will come as:

**Valid Boundary Conditions: Age = 30, 30.1, 149.9, 150**

**Invalid Boundary Conditions: Age = 29.9, 150.1**

Did you see the difference? We take the minimal acceptable value on either side of the boundary. If we take the value as 30.01, then we end up testing the software for two decimals where the requirement is one decimal place. It is a separate test condition and should not be mixed up with Boundary value.

Measurement of the Boundary coverage for a partition can happen as the number of boundary values tested divided by the total number of boundary test values identified.

**Q.13 what is Equivalence partitioning testing?**

Equivalence partitioning is a [black-box testing](https://testsigma.com/guides/black-box-testing/) technique that allows testers to group input data into sets or classes, making it possible to reduce the number of test cases while still achieving comprehensive coverage. This technique is particularly useful when dealing with a large range of input values.

Let’s walk through an example to understand the concept and process of Equivalence Partitioning.

**Testing a Login Form**

Suppose we have a login form with a username field. To apply Equivalence Partitioning, we can divide the possible input values into equivalence classes based on their expected behavior.

**Equivalence Classes:**

1. Valid Usernames: Alphanumeric usernames with a length of 5 to 15 characters.

2. Invalid Usernames: Usernames containing special characters or exceeding the length limit.

3. Empty Usernames: Leaving the username field empty.

**Step-by-Step Process of Equivalence Partitioning:**

1. Identify the input field: Determine the input field to be tested, such as the username field in our example.

2. Define equivalence classes: Categorize the possible input values into distinct equivalence classes. Each class represents a set of inputs with similar behavior or characteristics.

**3**. Determine representative values: Select representative values from each equivalence class. These values should cover the boundary conditions and critical scenarios within each class.

4. Create test cases: Generate [test cases](https://testsigma.com/guides/what-is-test-case/) that cover each equivalence class. For example, one test case would include a valid username, another would include an invalid username, and a third test case would cover an empty username.

5. Execute the test cases: Run the test cases using the defined equivalence classes to validate the behavior of the system. Observe and record the results for each test case.

By following this step-by-step process, testers can effectively apply Equivalence Partitioning to streamline test case design and achieve comprehensive test coverage while minimizing redundancy.

**Example of Equivalence Partitioning Technique:**

**I. Example – Testing User Registration Form**

In this example, let’s consider a user registration form that requires input for the age field. The Equivalence Partitioning technique can be applied to divide the possible input values into different equivalence classes.

**Equivalence Classes:**

**Valid Age: 18-60 (inclusive)**

**Invalid Age:** **Below 18 and above 60**

**Empty Age Field**

**Implementation with Test sigma:**

1. Using Test Sigma’s intuitive interface, create a new test case for the user registration form.
2. Add a test step to enter a valid age within the range of 18-60. This step represents the Valid Age equivalence **process.**
3. Add another test step to enter an invalid age below 18 or above 60. This step represents the Invalid Age equivalence class.
4. Include a test step to leave the age field empty. This step represents the Empty Age Field equivalence class.
5. Run the test case in Test Sigma’s automation environment, which will execute the test steps based on the equivalence classes defined.
6. Test Sigma will generate comprehensive test reports, highlighting the results for each equivalence class, allowing you to identify any issues or defects

**Advantages of Equivalence Partitioning:**

1. **Enhanced Test Coverage:** Equivalence partitioning allows testers to cover a wide range of input values by selecting representative test cases from each equivalence class, ensuring comprehensive coverage while minimizing redundant testing.
2. **Efficiency in Test Case Design:** By dividing input values into equivalence classes, test case design becomes more structured and systematic. It helps testers identify critical and boundary values for each class, enabling focused testing on potential areas of failure.
3. **Time and Effort Savings:** Equivalence partitioning helps optimize testing efforts by reducing the number of test cases needed while still maintaining sufficient coverage. This leads to time and effort savings, making the testing process more efficient.
4. **Defect Detection:** Equivalence partitioning increases the likelihood of detecting defects by targeting specific equivalence classes and their boundaries. By testing representative values from each class, testers can identify potential issues or failures.

**Disadvantages of Equivalence Partitioning:**

1. **Limited to Input Values:** Equivalence Partitioning primarily focuses on input values and their equivalence classes. It may not address other factors, such as system behavior or interaction between components, which can also contribute to defects.
2. **Complex Scenarios:** In complex scenarios with multiple inputs or dependencies, defining accurate and comprehensive equivalence classes can be challenging. It requires careful analysis and consideration of various factors, potentially increasing the complexity of the [**testing process**](https://testsigma.com/blog/principles-of-software-testing-2/).

1. **Requirement for Domain Knowledge:** Equivalence Partitioning relies on domain knowledge and understanding of the system under test. Testers need to have a clear understanding of the input values and their classifications to accurately identify and define equivalence classes.
2. **Potential for Overlooking Defects:** While Equivalence Partitioning improves test coverage, it does not guarantee the detection of all defects. There is a possibility of overlooking specific scenarios or edge cases that fall outside the defined equivalence classes.

**Q.14 what is integration Testing?**

* Integration testing is a software testing phase that focuses on verifying the interactions and interfaces between integrated components or systems.
* This testing occurs after individual modules have been unit tested and aims to ensure that they work together as intended.
* By combining modules, integration testing identifies issues related to data flow, communication protocols, and external dependencies, which might not be apparent during unit testing. The primary goal is to detect interface defects and integration errors that could affect the application’s overall functionality.
* There are several approaches to integration testing, including big bang, incremental, and top-down or bottom-up strategies.
* In the big bang approach, all components are integrated simultaneously and tested as a whole, while the incremental method integrates and tests components step by step.
* Each approach has its advantages, depending on the project’s complexity and the development process.
* By implementing integration testing, development teams can ensure that different parts of the application collaborate effectively, leading to a more reliable and efficient final product.

**Why is Integration testing important?**

Integration testing plays a critical role in the Software Development Life Cycle (SDLC) by ensuring that the various components of an application work seamlessly together. This phase of testing helps identify defects that may arise from the interactions between integrated modules, thereby improving overall software quality.

**Some key reasons for conducting integration testing are discussed below:**

* Early Detection of Issues: Integration Testing helps find the issues early on saving time, efforts and costs.  
  Example: If a payment processing module and an order management system are integrated, integration testing can reveal issues such as incorrect data transfer or communication failures between these components early in the development cycle, preventing costly fixes later.
* Validation of Interactions: It helps validate interactions between different individual modules ensure seamless communication for better functionality.  
  Example: In a social media application, integration testing can validate how the user profile module interacts with the messaging and notification modules, ensuring that updates in one system reflect correctly in others.
* Improved Software Quality: It helps see the overall picture and consolidated functionalities for enhanced software quality.  
  Example: An e-commerce website may have multiple modules like inventory, payment, and user accounts. Integration testing ensures these modules interact correctly, enhancing the overall quality and reliability of the application.
* Enhanced User Experience: Integration Testing focuses on delivering a seamless user experience without any glitch.  
  Example: In a mobile banking app, integration testing can ensure that features like fund transfers and transaction notifications work smoothly together, providing a seamless user experience.
* Identification of Interface Defects:  
  Example: If an API used to fetch data from an external source fails during integration testing, it can be identified before it impacts end-users, thereby ensuring data integrity across the application.
* Facilitates Continuous Integration:  
  Example: In Agile environments, where continuous integration is practiced, running integration tests regularly ensures that new code changes do not break existing functionality, maintaining application stability throughout development.
* Integration testing is vital for ensuring that individual components of a software application interact correctly, significantly enhancing the quality and reliability of the final product.
* By detecting issues early, validating interactions, and improving user experiences, integration testing becomes an indispensable part of the SDLC.
* It helps development teams build robust software that meets user expectations and minimizes the risk of defects arising from component integration.

**Purpose of Integration Testing**

The main purpose of integration testing is to validate that different software component, subsystems, or applications work together as a system to achieve the desired functionality and performance. Integration testing helps to identify and resolve any issues that may arise when components are combined, such as compatibility issues, performance problems, incorrect communication, or data corruption.

**Here are some specific objectives of integration testing:**

1. Verify the interactions between components.
2. Ensure compatibility.
3. Detect problems early.
4. Improve the overall reliability of the system.
5. Improve the quality of the system by identifying and fixing issues before they become more difficult and expensive to resolve.

The purpose of integration testing is highly important in the overall software development life cycle.

### Types of Integration Testing

There are several types of integration testing that can be performed to test the interactions between software components:

1. **Big-bang integration testing**: This integration testing involves integrating all the components at once and testing them as a complete system. The method is typically used when the components are relatively independent and can be tested individually.
2. **Top-down integration testing**: You can use top-down integration testing when the components are integrated and tested from the highest level to the lowest level. The approach is used when the higher-level components depend on the lower-level components.
3. **Bottom-up integration testing**: The integration testing type involves integrating and testing the components from the lowest level to the highest level.
4. **Sandwich/hybrid integration testing**: This integration testing involves combining elements of both top-down and bottom-up integration testing.
5. The components are tested from both the top and bottom levels, with stubs and drivers used to simulate the missing components.
6. **Continuous integration testing**: It involves continuously integrating and testing the components as they are developed. The method helps to catch and resolve problems early in the *development process, improving the overall quality of the system.*

The type of integration testing to be used depends on the specific requirements of the software system and the development process.

**Integration Testing Techniques**

Integration testing is a crucial phase in the software development process, focusing on validating the interactions between integrated components of an application. Various testing techniques can be employed to ensure that these components work together correctly, each with its own strengths and use cases.

These techniques can be categorized into three main approaches: [Black Box](https://www.browserstack.com/guide/black-box-testing), [White Box](https://www.browserstack.com/guide/white-box-testing), and [Grey Box testing](https://www.browserstack.com/guide/grey-box-testing).

**Black Box Testing Techniques**

1. **State Transition Technique**:
   * **Description**: This technique is used to test the application’s behavior under various states and transitions between those states.
   * **Example**: In a banking application, a user may transition from a “Logged Out” state to a “Logged In” state after entering credentials. State transition testing can verify that all actions (like accessing account information) work correctly in both states.
2. **Decision Table Technique**:
   * **Description**: This technique utilizes a table to represent combinations of inputs and their corresponding outputs, facilitating testing of complex business rules.
   * **Example**: For an online shopping site, a decision table can be created for various payment methods (credit card, PayPal, etc.) and their outcomes based on different user scenarios, ensuring that all conditions are tested.
3. **Boundary Value Analysis**:
   * **Description**: This technique focuses on testing the boundaries of input ranges to identify errors at the extremes.
   * **Example**: If a form requires an age input between 18 and 60, boundary value analysis would test the values 17, 18, 60, and 61 to ensure the system correctly handles these edge cases.
4. **All-Pairs Testing**:
   * **Description**: This combinatorial testing technique aims to test all possible pairs of input parameters to uncover interaction defects.
   * **Example**: In a software that allows users to select options like color and size, all-pairs testing would verify combinations like red-small, red-large, blue-small, and blue-large.
5. **Cause and Effect Graph**:
   * **Description**: This technique uses a graphical representation to map causes (inputs) to their effects (outputs) for systematic testing.
   * **Example**: In an email application, actions like sending, receiving, or deleting emails can be mapped out to ensure each cause produces the expected outcome.
6. **Equivalence Partitioning**:
   * **Description**: This technique divides input data into equivalent partitions to reduce the number of test cases while still covering all scenarios.
   * **Example**: For a user registration form that requires a username between 5 to 15 characters, equivalence partitions would include inputs like “abc” (invalid), “abcdef” (valid), and “abcdefghijklmnop” (invalid).
7. **Error Guessing**:
   * **Description**: This technique relies on the tester’s intuition and experience to guess potential error-prone areas in the application.
   * **Example**: A tester might focus on input fields that are known to cause issues, like special characters in a username field, to see if the system handles them correctly.

**White Box Testing Techniques**

1. **Data Flow Testing**:
   * **Description**: This technique focuses on the lifecycle of data variables, ensuring that data is properly defined, used, and cleared.
   * **Example**: In a payroll system, data flow testing would verify that employee records are correctly updated after a salary change.
2. **Control Flow Testing**:
   * **Description**: This technique examines the control flow of the program to ensure that all paths through the code are executed.
   * **Example**: In an e-commerce application, control flow testing would ensure that different user paths—such as browsing products, adding to cart, and checking out—are all correctly executed.
3. **Branch Coverage Testing**:
   * **Description**: This technique checks that every branch (true/false paths) in the code is executed at least once during testing.
   * **Example**: If a function checks if a user is eligible for a discount based on age, branch coverage testing would ensure that both the eligible and non-eligible paths are tested.
4. **Decision Coverage Testing**:
   * **Description**: This technique ensures that every decision point in the code is tested for both true and false outcomes.
   * **Example**: In a login system, decision coverage testing would verify that both successful and failed login attempts are handled correctly.

**Grey Box Testing Techniques**

Grey Box testing combines elements of both Black Box and White Box testing, leveraging knowledge of the internal structure while focusing on the output.

**Integration Testing with Internal Interfaces**:

* **Description**: Testers validate the interactions between components while being aware of the internal workings of the modules.
* **Example**: A tester might verify the integration between a front-end interface and a back-end database, ensuring data is correctly transferred and processed while understanding the underlying database structure.
* Integration testing is essential for verifying that individual components of a software application work together as intended.
* Utilizing a variety of techniques—Black Box, White Box, and Grey Box—can significantly enhance the effectiveness of integration testing.
* Each technique offers unique benefits and can be selected based on the specific requirements and complexities of the application.
* By systematically applying these techniques, development teams can identify defects early in the SDLC, ensuring a more robust and reliable final product.

**Q.15 what determines the level of risk?**

* The accepted foundation of any system of managing security is risk management. Risk management is the coordinated activities of the company to direct and control security in relationship to threats and vulnerability that may influence the objectives of the company.

**Risk Categories**Assists in sorting and organizing; examples include:

a. ***Safety:***Includes life and health exposures arising out of university operations, events, and activities.

b. ***Strategic*:**Exposure to uncertainty related to long-term policy directions of the institution—the “big picture” risks.

c. ***Financial*:**Exposure to uncertainty regarding the management and control of the financial resources and physical assets of the institution.

d. ***Operational*:**Exposure to uncertainty related to day-to-day business activities.

e. ***Reputational:***Exposure to uncertainty related to brand, perceived value, organizational status, and public perception and trust. Employee and educational integrity and a clear statement of the ethics and moral values emanating from the top are important components of this risk.

f. ***Compliance:***Includes external regulatory agencies, internal and external reporting, financial and non-financial information. The Risk Assessment starts once the scope of the security policy is established, and begins with

1. Asset configuration or Identification of Assets, with a listing of
   1. user name and
   2. Contact details. then establish a
2. Confidentiality classification which is the level of confidentiality the asset requires. This is followed by Establishing the
3. Impact Range and
4. Likelihood Range with low values meaning low impact and low vulnerability. You then need to decide your
5. Acceptance Criteria and that established your
6. Risk Assessment Scope. SO now you can establish your
7. Risks, which is a combination of
   1. Threats and
   2. Vulnerabilities, which have Impact value assignment and Likelihood Level assignment.

* This becomes your **Risk Level** (Accepted and Residual Risks) and everything below the Risk Level is Accepted Risk which requires no further action.
* Residual Risks are then treated in several ways, but I am going to stop here because the answer is now provided.
* This is an actual sequence of events with some technical and owner, and responsibilities missing.
* This process make take a concentrated effort with technical applications used to do metrics and keep the data being generated as this assessment process should be done when any and all changes to the system take place, including incidents and asset changes.

**Q.16 what is Alpha Testing?**

* Alpha testing is an early phase of software testing where the internal development team checks the software for bugs and issues before it is released to external users.
* It aims to ensure that the software functions correctly and meets user requirements in a controlled environment.
* Alpha testing falls under the category of Acceptance Testing. Some unit or smoke tests of features may have been performed during the development of the software, however it is important to Alpha test the overall completed product in order to catch all possible issues and give the developers a chance to address them before real user tests are carried out.
* Alpha testing is done by internal developer and QA teams, with its main goal being to ensure that the software is functional, reliable, and free of any defects or errors.
* Alpha testing generally attempts to simulate real user behavior by using black and white box testing techniques.
* Black box testing is done by not taking into consideration the internal workings or algorithm of the software being tested.
* This kind of test simply evaluates whether the software is behaving as it should, by entering inputs that a real user might and evaluating whether the outputs are as expected.
* White box testing on the other hand is generally performed by testers who have thorough knowledge of the internal code of the software.
* This kind of testing looks at the structure, code, design, and integrity of the software in order to identify points of potential improvement with regards to security, design, functionality, or usability.

**Objectives of Alpha Testing**

Understanding what you’re looking for and why is crucial during alpha testing. If the results of this testing are not actionable, it’s best to identify this early in the development cycle. Doing so can save time and money by preventing unnecessary tests later on.

Some of the objectives include:

1. **Identify Bugs**: The main goal is to find and fix defects in the software before it goes to users.
2. **Ensure Functionality**: Alpha testing checks that all features work as intended and meet the requirements.
3. **Evaluate Usability**: It assesses how user-friendly the application is, ensuring a good experience for users.
4. **Verify Performance**: The testing evaluates the software’s speed and stability under different conditions.
5. **Gather Feedback**: Feedback from testers helps developers make improvements and refine the product before the beta phase.
6. **Prepare for Beta Testing**: Alpha testing aims to resolve major issues, making the software ready for the next stage of testing with a wider audience.

**Who performs Alpha Testing?**

The alpha testing process involves a small group of individuals closely associated with the project who work together to evaluate the software in a controlled environment. These Alpha testers are typically classified as **white box** or **black box** testers.

**White box testers** are technical team members, such as developers, with deep knowledge of the code and an understanding of how the software should function.

In contrast, **black box testers** are non-technical individuals who focus on real-world scenarios and user experiences. They may be internal employees outside the development team or a select group of users who are given early access to the software in exchange for their feedback, ensuring a thorough evaluation of the software before it moves on to beta testing.

**When is Alpha Testing Done?**

Alpha testing is the last phase of internal testing. Alpha testing is more thorough than all previous tests, and tests the overall software product end-to-end.

It is generally carried out right after software development before the software is handed over for Beta testing by real users external to the company. It is called Alpha testing since it precedes the Beta testing phase.

**Advantages of Alpha Testing**

**Core advantages include:**

1. **Crucial Insights: Provides** valuable information about the software’s reliability and potential issues.
2. **Early Feedback**: Provides early feedback that aids in improving product quality.
3. **Resource Allocation**: Allows the team to focus on other projects by addressing major concerns early.
4. **User-Centric Design**: Enables developers to gather user feedback, benefiting the design process and guiding feature optimization for stakeholders.
5. **Enhanced Strategy**: Improves testing strategies, especially when using tools like Playwright Test, leading to better results.
6. **Team Confidence**: Helps the software team gain confidence in their product before market release.

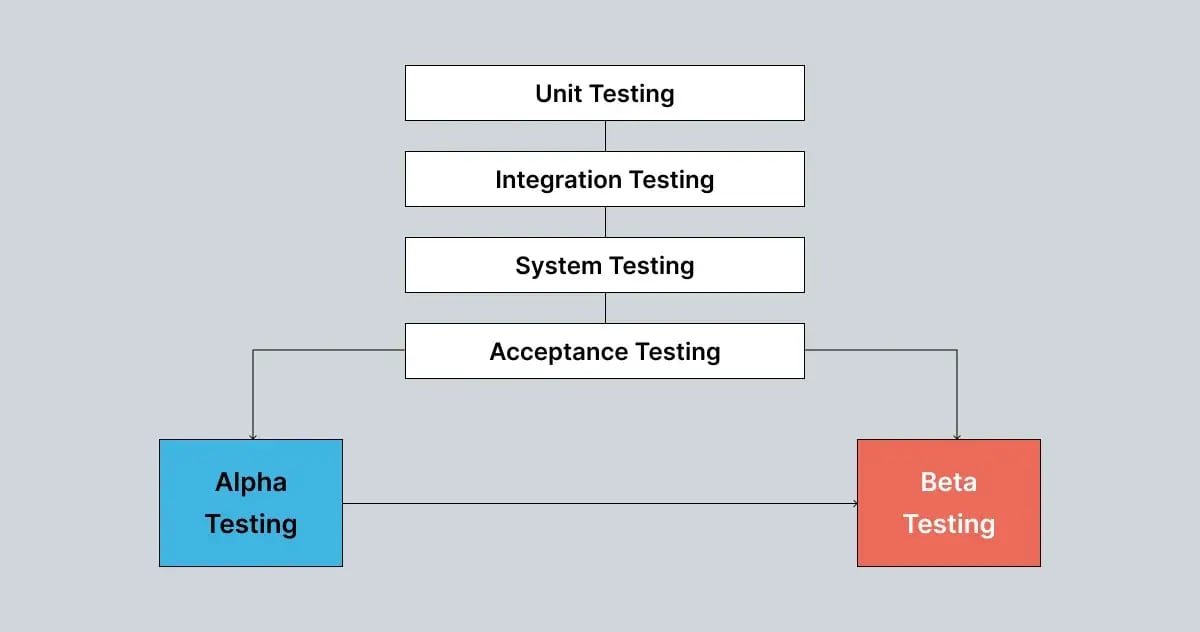
**Limitations of Alpha Testing**

**Some limitations include:**

1. **Undetected Defects**: Some defects may remain undiscovered, as alpha testing primarily focuses on user reactions rather than exhaustive defect detection.
2. **Time-Consuming for Large Projects**: For larger projects already subjected to thorough testing, alpha testing can be time-consuming and may delay release due to the need for extensive test planning and documentation.
3. **Environmental Issues**: While it mimics the production environment, some defects may arise from environmental factors not present in the alpha testing phase.
4. **Not Always Necessary**: For small projects, alpha testing may not be required, as time and budget constraints can make it inefficient.

**Q.17 what is Beta Testing?**

Beta testing is a critical phase of [acceptance testing](https://www.globalapptesting.com/blog/types-of-qa-testing) in which end-users evaluate a product before its official release. The beta tester group is typically smaller than the full customer base expected after the product launch.  
  
This phase offers developers valuable insights into the product's performance and usability from the end-user's perspective, which often differs significantly from the developer's.   
  
Feedback from beta testers is collected, analyzed, and used to [refine and enhance the product](https://www.globalapptesting.com/blog/quality-assurance-vs-quality-control), ensuring it meets the users' needs and expectations before the official launch.



In simple terms, when end users tell you your product is great, you know it's genuine feedback, not just your own bias. Although beta testing depends on real users and can't be fully automated, automation tools can help manage the process.

What are the types of beta testing?

There are five main types of beta testing, each helping to [ensure your software is top-notch](https://www.globalapptesting.com/blog/how-do-you-ensure-quality-in-the-software-you-create). While they all aim to improve the product, they do so in different ways.

1. Private/Closed Beta testing

Closed beta testing involves releasing the software to a select group of users who test its features and functionalities. The number of testers is limited and chosen carefully to meet specific needs. Closed betas are great for testing particular aspects, like a new app feature or a website's landing page.

2. Public/Open Beta testingOpen beta testing, or public beta testing, is open to a larger number of testers. This approach allows developers to gather extensive feedback on how the app performs and how users interact with it. Open betas help assess the app's scalability and infrastructure by exposing it to a wider audience.



3. focused Beta testing

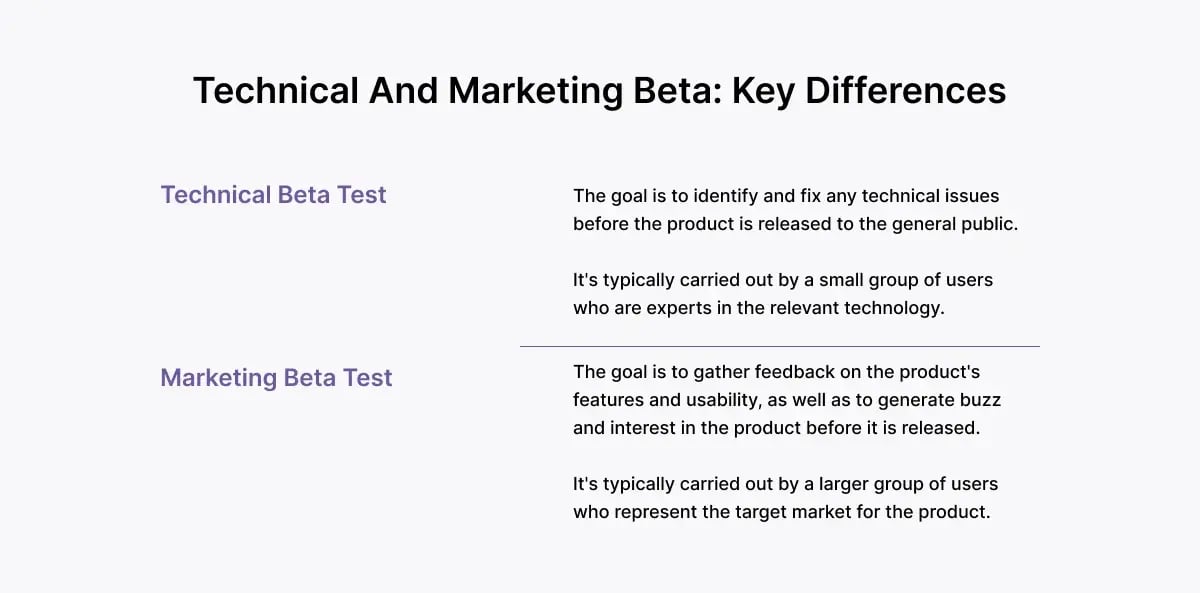
Focused beta testing targets feedback on a specific feature, often when new features are introduced to existing products. Depending on the developers ' needs, this approach can be either closed or open and helps ensure that new features are well-integrated and functional.

4. Technical Beta testing

Technical beta testing is a subtype of closed beta testing involving tech-savvy testers who look for complex bugs. This type of testing requires testers with the expertise to identify and report intricate issues, making it more controlled than open beta testing.

5. Marketing Beta testing

Marketing beta testing aims to generate media attention for a new app. It evaluates both the product and the effectiveness of marketing channels in gaining recognition. This type of testing helps gauge potential customer interest and predict retention rates. If interest wanes before the official release, it signals the need for improvements to boost the app's appeal and retention.

  
  
Understanding these types of beta testing helps developers choose the best method for their needs, ensuring the product is thoroughly tested and refined before its official launch.

**Q.18 what is component Testing?**

**Component Testing**

Component Testing is a type of software testing where individual components or modules of a software application are tested in isolation to verify their functionality, reliability, and performance. It is typically performed during the development phase and is often done by developers or testers.

**Key Aspects of Component Testing**

1. Scope: Tests a single component independently before integrating it with other components.
2. Objective: Ensures that the component functions correctly as per the requirements.
3. Performed By: Developers (unit testing) or testers (black-box testing).
4. Techniques Used:
   * Black-box testing: Focuses on inputs and expected outputs.
   * White-box testing: Tests internal logic and code structure.
   * Grey-box testing: Combines both black-box and white-box approaches.
5. Automation: Can be automated using tools like JUnit (Java), NUnit (.NET), PyTest (Python), Selenium, etc.

**Example of Component Testing**

Imagine +a login module of an application.

* Entering valid credentials → User should log in successfully.
* Entering incorrect credentials → Error message should be displayed.
* Leaving field’s blank → Validation messages should appear.

**Benefits of Component Testing**

1. Early defect detection  
2. Easier debugging  
 3.Improves code quality  
 4.Reduces overall cost of fixing defects

**Q.19 What is GUI testing?**

**GUI Testing (Graphical User Interface Testing)**

GUI Testing is a type of software testing that evaluates the **Graphical User Interface (GUI)** of an application to ensure that it meets the specified requirements and functions correctly. It focuses on **visual elements, user interactions, and usability** to ensure a smooth user experience.

**Key Aspects of GUI Testing**

1. **Visual Appearance** – Checks layout, fonts, colors, icons, buttons, and alignment.
2. **Functionality** – Ensures buttons, links, dropdowns, and other UI elements work as expected.
3. **Usability** – Verifies ease of use, navigation, and overall user experience.
4. **Consistency** – Ensures uniform design across different ++pages/screens.
5. **Error Handling** – Checks proper error messages and notifications.
6. **Responsiveness** – Ensures UI works on different screen sizes and resolutions.

**Examples of GUI Testing**

* **Login Page:**
  + Verify that the username and password fields accept valid inputs.
  + Check if the "Login" button is clickable.
  + Ensure error messages appear for incorrect login attempts.
* **E-commerce Website:**
  + Check if product images are displayed correctly.
  + Ensure the "Add to Cart" button adds the item correctly.
  + Verify that filters and sorting options work properly.

**GUI Testing Methods**

**Manual Testing** – Testers manually check UI elements.  
 **Automated Testing** – Tools like Selenium, TestComplete, or Appium automate UI testing.

**Enhances User Experience  
 Ensures Proper Functionality  
 Identifies Visual & Usability Issues  
 Improves Application Consistency**

**Q.20 what is Adhoc Testing?**

Adhoc Testing is an informal and unstructured type of software testing conducted without any predefined test cases or plans. The primary goal is to identify defects through random and exploratory testing. Testers rely on their experience, intuition, and domain knowledge to find bugs that may not be covered in formal testing.

Key Characteristics of Adhoc Testing

1. Unstructured & Improvised – No predefined test cases or documentation.
2. Exploratory in Nature – Testers use their intuition and domain knowledge.
3. Performed Without Planning – No formal test plan is required.
4. Effective in Finding Critical Bugs – Helps uncover defects missed in scripted testing.
5. Usually Done After Formal Testing – Used when time is limited and critical areas need quick validation.

Types of Adhoc Testing

1. Monkey Testing – Randomly interacting with the application to find unexpected crashes or errors.
2. Buddy Testing – Two testers (usually a developer and a tester) work together to test the application.
3. Pair Testing – Two testers collaborate to explore different parts of the application.

When to Use Adhoc Testing?

* When there is limited time for structured testing.
* After formal testing to check for missed defects.
* When testing an unstable or new feature.
* To simulate real user behavior.

Adhoc testing is best used alongside structured testing methods to improve test coverage.

**Q.21 what is Load Testing?**

Load Testing is a type of performance testing that evaluates how a software application behaves under expected user loads. The goal is to determine if the system can handle a specified number of concurrent users, transactions, or requests without performance degradation.

**Key Aspects of Load Testing**

1. **Measures System Performance** – Evaluates response time, speed, and resource usage.
2. **Identifies Bottlenecks** – Detects slowdowns, memory leaks, and database issues.
3. **Ensures Stability** – Confirms the system can handle peak loads efficiently.
4. **Prevents Downtime** – Helps avoid crashes or failures under heavy user traffic.

**Example Scenario**

Imagine an e-commerce website like Amazon. During a festive sale, thousands of users try to place orders simultaneously. Load testing ensures the website can handle high traffic without slowing down or crashing.

**Common Load Testing Tools**

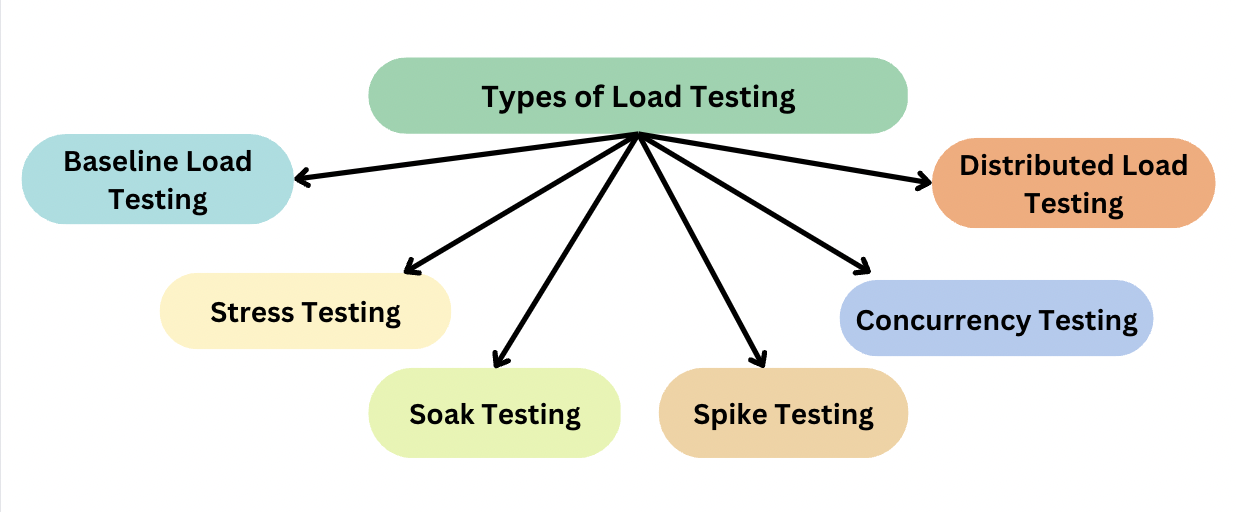
* **JMeter** – Open-source tool for web application performance testing.
* **LoadRunner** – Enterprise-grade tool for large-scale load testing.
* **Gatling** – Developer-friendly tool for automated performance testing.
* **Apache Bench (AB)** – Simple command-line tool for testing web servers.

**When to Perform Load Testing?**

* Before launching a **new website or application**.
* Before **high-traffic events** like sales or promotions.
* After major **code updates** to ensure stability.
* As part of **regular performance monitoring**.

**Different Types of Load Testing**

Load testing can be categorized into several different types, each with its specific focus and objectives. The main types of load testing include:



* **Baseline Load Testing**: Used to set a performance baseline for the system under normal or typical load conditions. These baselines can be used to find standards against which the system behavior can be observed.
* **Stress Testing**: This method pushes the system beyond its normal capacity to identify breaking points and failure conditions. It helps determine how much load the system can handle before it crashes.
* **Soak Testing**: By applying a steady load over an extended period, this test evaluates long-term performance. It helps detect memory leaks, resource depletion, and gradual performance degradation.
* **Spike Testing**: Abrupt and substantial surges in user traffic are simulated to observe the system’s response. This ensures the application remains stable during sudden increases in load.
* **Concurrency Testing**: Multiple users or processes are made to access the system simultaneously to measure how well it handles concurrent interactions. This helps assess responsiveness under high user engagement.
* **Distributed Load Testing**: Instead of relying on a single machine, multiple systems or virtual users are used to generate large-scale traffic. This approach allows testing at a much greater load capacity.

**Q.22 what is Stress Testing?**

Stress Testing is a type of performance testing that evaluates how a software application behaves under extreme conditions, such as high traffic, heavy data load, or limited system resources. The goal is to determine the system’s stability, reliability, and failure recovery capabilities under stress.

Key Aspects of Stress Testing

1. Pushes the System Beyond Limits – Tests beyond normal load conditions.
2. Identifies Weak Points – Detects memory leaks, crashes, and slowdowns.
3. Evaluates System Recovery – Checks how well the system recovers after failure.
4. Simulates Real-World Scenarios – Helps prepare for unexpected traffic spikes.

Example Scenario

Imagine an online ticket booking system during a flash sale. If millions of users try to book tickets at the same time, stress testing ensures the system doesn’t crash and can recover smoothly if it does.

Types of Stress Testing

* Spike Testing – Sudden increase in users or load.
* Soak Testing (Endurance Testing) – Sustained high load over an extended period.
* Breakpoint Testing – Finding the exact point where the system fails.
* Failover Testing – Checking system recovery after failure.

Common Stress Testing Tools

* JMeter – Open-source tool for simulating high loads.
* LoadRunner – Enterprise solution for stress testing.
* Gatling – High-performance load testing tool.
* Locust – Python-based load testing tool for scalability tests.

When to Perform Stress Testing?

* Before major releases to check stability.
* When expecting sudden user spikes (e.g., festivals, promotions).
* To verify disaster recovery plans.

**Types of Stress Testing**

There are several types of stress testing in software testing, each focusing on different aspects of the system:

1. **Application Stress Testing:** Focuses on stressing the application itself.
2. **System Stress Testing**: Tests the entire system, including hardware and software components.
3. **Distributed Stress Testing**: Simulates stress from multiple sources simultaneously.
4. **Transactional Stress Testing**: Stresses the system with a high volume of database transactions.
5. **Systemic Stress Testing**: Stresses multiple interconnected systems. This type of testing is often performed in conjunction with [*system integration testing*](https://www.f22labs.com/blogs/system-integration-testing-sit-a-complete-guide/) to ensure the entire system performs well under stress.

**Method for Stress Testing :**

1. **Prepare/Plan to Test:**Determine the objectives, goals, metrics, and required testing environment. Understanding how to set up an effective [*test environment*](https://www.f22labs.com/blogs/test-environment-in-software-testing-a-beginners-guide/) is crucial for accurate stress testing results.
2. **Create Test Scenarios:** Design scenarios that will push the system beyond its normal operational capacity.
3. **Configure Tools for Testing:** Install [*Apache Jmeter*](https://www.f22labs.com/blogs/mastering-performance-testing-with-jmeter-a-comprehensive-guide/), LoadRunner, or Gatling. Configure to simulate very high loads.
4. **Run Tests:** Perform stress tests. Ramp loads on the system that will eventually break the system.
5. **Log and Collect Data:** Logs how the system reacts and collects performance metrics throughout the test.
6. **Analyze Results:** Analyzes data to identify bottlenecks, failure points, and areas for optimization.
7. **Report and Recommend:**The findings recorded with recommendations for system optimizations.

**Best Practices for Stress Testing**

To get the most out of stress testing, consider these best practices:

1. **Start Early**: Incorporate stress testing early in the development cycle to catch issues before they become costly.
2. **Use Realistic Scenarios**: Base your stress tests on real-world scenarios that your system might encounter.
3. **Vary Test Parameters**: Don't just increase user load; vary other factors like data size, network conditions, etc.
4. **Monitor Resource Utilization**: Keep an eye on CPU, memory, disk I/O, and network usage during tests.
5. **Test Regularly**: As your system evolves, regular stress testing helps maintain reliability.
6. **Automate When Possible**: Automation allows for more frequent and consistent stress testing. Familiarising yourself with various [*types of automation testing*](https://www.f22labs.com/blogs/10-types-of-automation-testing-you-need-to-know/) can help you choose the best approach for your stress testing needs.

**Few Challenges of Stress Testing**

**Challenge 1: Setting up a test environment is not easy**

It is challenging to set a test environment that closely mimics the production environment because of differences in hardware configurations, software setups, and data.

Solutions:

* **Use Virtualization and Containers:** Software like Docker, Kubernetes, and Vagrant can be used in creating encapsulated, controlled environments that share many properties with the production environment.
* **Testing at Cloud Platform:**The cloud platforms AWS and Azure offer scalable environments that share most properties of the production environment, thus making configurations and tests easy to configure and execute.
* **Automate Setup:**An environment will be similarly created to that of a production environment with the help of Terraform and Ansible .
* **Simulate Data:**With the aid of tools like Mockaroo, test data can be generated that could pass for real or actual data.

**Challenge 2: Selection of Testing Tools**

It is difficult to determine which testing tool is appropriate for a project. Every project has different needs based on testing requirements.

Solution:

* **Requirements-Based Evaluation:** The system requires identifying what is needed (for example functional, load, security testing) and therefore selects the specific tool needed.
* **Trail Multiple Tools:**Try using the free version or open-source tools before deciding to buy it.
* **Seek Experts:** Get recommendations from the experts and peers through industry forums and communities
* **Scalable tools:**Scale up the ones that integrate well with the current systems, CI/CD pipelines, and ready to adapt to increased processing power

**Challenge 3: Resource Intensive Stress Testing**

The needed computing power is very high and expensive in stress testing

Solutions:

* **Leasing Really Powerful Servers:**Using cloud resources allow you to lease really powerful servers temporarily for stress testing, thereby avoiding significant long-term costs.
* **Testing Across Multiple Systems:** You can use tools such as JMeter or Locust to distribute the load across multiple systems. This can help prevent the overloading of one machine.
* **Test Only the Important Ones:**Instead of testing all the features of the application at all times, only test the important ones, whether it is a high-traffic area of the application or not.
* **Pre-Provisioned Services:** Use cloud services like Blaze Meter or Loader.io, built for stressing so you are not creating your infrastructure.

**Challenge 4: Analysis of Stress Test Reports**

Analysis of outcomes received from a stressing is tricky and needs some technical knowledge.

Solutions:

* **Reporting Tools:**New Relic, Dynatrace, and Grafana, enable you to visualize the test reports so you can easily find issues without needing deep technical knowledge.
* **Hire or Train Experts:** Hire performance engineers or train your team on tools like JMeter, or obtain certifications on performance testing.
* **Automated Insights:**New tools such as Neo, Gremlin (etc.) rely heavily on AI to help interpret the result of the test and provide actionable insights.

**Collaborate with Developers:**Engage with the development team to understand the performance goals and how to convert

**Q.23 Mention what big bang testing?**

Big Bang Testing is an **integration testing** approach where all the individual components or modules of a system are combined and tested as a whole, rather than integrating them step by step.

**Key Features of Big Bang Testing**

1. **All Modules Integrated Simultaneously** – No incremental testing.
2. **Executed After Full System Development** – Performed at the end of the development phase.
3. **Useful for Small Systems** – Suitable when modules are not dependent on each other.
4. **Detects Major Issues Late** – Since everything is tested together, identifying specific defects is difficult.

**Example Scenario**

Imagine a **banking application** with modules like **Login, Fund Transfer, Account Details, and Transaction History**. In Big Bang Testing, all these modules are integrated at once and tested together instead of testing them individually.

**Advantages of Big Bang Testing**

**1. Quick Testing – Everything is tested at once.  
 2. Good for Small Systems – Works well when there are fewer modules.**

**Disadvantages of Big Bang Testing**

1. **Difficult to Debug** – If an issue is found, pinpointing the exact module causing the problem is hard.  
2. **High Risk** – Since testing happens late, major defects might go unnoticed until the end.  
3. **Time-Consuming** – If many bugs are found, fixing and retesting take longer.

**Q.24 what is the Purpose of Exit-Criteria?**

**Exit Criteria** define the conditions that must be met before stopping a testing phase. It ensures that testing is completed effectively and that the software is ready for the next phase or release.

**Key Purposes of Exit Criteria**

1. **Ensures Testing Completeness**
   * Confirms that all planned test cases have been executed.
   * Validates that critical defects are resolved.
2. **Defines Quality Standards**
   * Ensures the software meets the required quality benchmarks.
   * Verifies that performance, functionality, and security are acceptable.
3. **Prevents Incomplete Releases**
   * Avoids releasing software with critical issues.
   * Reduces the risk of failure in production.
4. **Helps in Decision Making**
   * Assists stakeholders in deciding whether the software is ready for deployment.
   * Provides data-driven insights on testing effectiveness.
5. **Saves Time and Resources**
   * Prevents unnecessary over-testing.
   * Helps optimize testing efforts by defining a clear stopping point.

**Common Exit Criteria in Testing**

**1Test Case Execution** – A certain percentage (e.g., 95%) of test cases must pass.  
 2. **Defect Rate** – No high-severity or critical defects should be open.  
 3. **Performance Benchmarks** – The application must meet speed and stability requirements.  
**4. Code Coverage** – A minimum percentage of code (e.g., 85%) must be tested.  
**6. Customer Requirements** – All functional and non-functional requirements should be validated.

**Q.25 When should “Regression –Testing “be performed?**

Regression Testing should be performed whenever there are changes in the software to ensure that the existing functionalities still work as expected. It helps detect unintended side effects caused by code modifications.

**Key Scenarios for Performing Regression Testing:**

1. **After Fixing a Bug**
   * To verify that the bug fix did not break other functionalities.
2. **After Adding a New Feature**
   * To check if the new feature has affected existing features.
3. **After Code Optimization or Refactoring**
   * To ensure performance improvements did not introduce new defects.
4. **After Changes in Configuration or Environment**
   * To validate that the application works correctly on different browsers, OS, or databases.
5. **Before a Major Release**
   * To confirm that all previous functionalities are intact before deployment.
6. **During Continuous Integration (CI/CD)**
   * Automated regression testing should be part of the CI/CD pipeline to catch defects early.

**Types of Regression Testing:**

1. **Full Regression Testing** – Performed when there are major code changes.  
 2. **Partial Regression Testing** – Focuses only on the affected modules.  
 3. **Unit Regression Testing** – Performed at the code level to test individual components.

**Regression Testing Tools:**

* **Selenium** – For automated web application testing.
* **JUnit, TestNG** – For Java-based regression testing.
* **Appium** – For mobile application regression testing.
* **Jenkins** – For CI/CD pipeline integration with automated regression testing.

**Q.26 Explain types of performance testing?**

Performance Testing is conducted to evaluate the speed, responsiveness, and stability of a software application under different conditions. Below are the major types of performance testing:

**1. Load Testing**

✅ **Purpose:** Determines how the system behaves under an expected number of concurrent users or transactions.  
✅ **Example:** Checking how an e-commerce website performs with **1,000 simultaneous users**.  
✅ **Tools Used:** JMeter, LoadRunner, Gatling

**2. Stress Testing**

✅ **Purpose:** Tests system behavior under extreme conditions or beyond its capacity.  
✅ **Example:** Simulating **1 million users** on a website to check if it crashes.  
✅ **Tools Used:** JMeter, LoadRunner, Locust

**3. Soak Testing (Endurance Testing)**

✅ **Purpose:** Evaluates system stability over an extended period under normal load.  
✅ **Example:** Running an application continuously for **24 hours** to check memory leaks.  
✅ **Tools Used:** JMeter, LoadRunner

**4. Spike Testing**

✅ **Purpose:** Examines system response to sudden and extreme spikes in traffic.  
✅ **Example:** A ticket booking site facing **10x traffic during a movie release**.  
✅ **Tools Used:** JMeter, LoadRunner

**5. Scalability Testing**

✅ **Purpose:** Determines if the system can scale up/down efficiently when load increases or decreases.  
✅ **Example:** Checking if an **online learning platform** can handle **growing users** during peak hours.  
✅ **Tools Used:** JMeter, Gatling

**6. Volume Testing (Flood Testing)**

✅ **Purpose:** Evaluates system performance by processing a large volume of data.  
✅ **Example:** Uploading **millions of records** into a database to check response time.  
✅ **Tools Used:** JMeter, Load Runner

**Key Benefits of Performance Testing**

1. Prevents slowdowns and crashes.  
 2. Improves user experience.  
 3. Ensures system reliability under different conditions.

**Q.27 Difference between Smoke Testing and Sanity Testing?**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Smoke Testing** | **Sanity Testing** |
| **Purpose** | This testing aims to confirm that the recently made build is steady enough to perform further rigorous testing and that the basic functionalities are working properly. | The goal is to test whether the bugs are fixed after the stable build and determine the system’s correctness. It just includes the module on which code changes take place. |
| **Focus of Testing** | Smoke testing verifies the basic functionalities of the entire system. | Sanity testing focuses on specific components where code changes have occurred. |
| **Methodology** | This testing can be performed in two ways- [manually](https://www.browserstack.com/guide/manual-testing-tutorial) and [automatically](https://www.browserstack.com/guide/benefits-of-automation-testing?searchUuid=58b6c436-2b73-4fc3-8807-8ff913ddd8d1). | This testing can be done without any [test cases](https://www.browserstack.com/guide/how-to-write-test-cases-for-login-page) or [test scripts](https://www.browserstack.com/guide/test-case-vs-test-script). |
| **Software Stability** | Smoke testing can make the software stable or unstable. This testing is done after every new build is released, including an end-to-end system verification, so in-depth testing is necessary. | The software should be comparatively stable for this test, which is done to verify a specific component, such as a newly introduced feature. So, in-depth testing is not possible for this. |
| **Issue Detection** | Executing smoke testing helps to ensure that the issues fixed on the previous build do not bother the vital functionalities of the application. | Executing sanity testing helps to save unnecessary testing effort and time because it’s only performed on some specific functionalities. |
| **Documentation** | Test documents and scripts are made for future reference. | There is no test document or test script needed for this testing. So, no future reference will be available. |
| **Relation to Other Testing Types** | It is assumed as a subset of [acceptance testing.](https://www.browserstack.com/guide/user-acceptance-testing) | Assumed as a subset of [regression testing.](https://www.browserstack.com/guide/regression-testing) |
| **Execution Timing** | Typically executed after every new build to ensure basic functionalities work. | Conducted after changes are made to specific modules to verify [bug fixes](https://www.browserstack.com/guide/hotfix-vs-bugfix). |

**Q.28 Difference between Priority and Severity?**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Severity in Testing** | **Priority in Testing** |
| **Definition** | Severity is a term that denotes how severely a defect can affect the functionality of the software. | Priority is a term that defines how fast we need to fix a defect. |
| **Parameter** | Severity is basically a parameter that denotes the total impact of a given defect on any software. | Priority is basically a parameter that decides the order in which we should fix the defects. |
| **Relation** | Severity relates to the standards of quality. | Priority relates to the scheduling of defects to resolve them in software. |
| **Value** | The value of severity is objective. | The value of priority is subjective. |
| **Change of Value** | The value of Severity changes continually from time to time. | The value of Priority changes from time to time. |
| **Who Decides the Defect** | The testing engineer basically decides a defect’s severity level. | The product manager basically decides a defect’s priority level. |
| **Types** | There are 5 types of Severities: Cosmetic, Minor, Moderate, Major, and Critical. | There are 3 types of Priorities: High, Medium, and Low. |

**Q.29 what is the difference between STLC and**

**SDLC?**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **SDLC** | **STLC** |
| **Origin** | Development Life Cycle | Testing Life Cycle |
| **Definition** | SDLC produces a superior-quality system that exceeds or meets users’ expectations, works excellently and competently in the current, planned, and strategic information technology infrastructure, and is lucrative to manage. | STLC, on the flip side, identifies what test actions to perform and when to achieve those test activities. Although tests distinct between Organizations, there is a specific test life cycle. |
| **Focus** | On both dev (development) as well as the test process. | On merely testing process. |
| **Performed** | The stages of the SDLC are ended before those of the Software Testing Life Cycle (STLC). | The stages of the Software Testing Life Cycle (STLC) are performed after the stages of SDLC. |
| **Objective** | All through the SDLC procedure, the aim is to overcome any obstacle on the way to effective/ successful software development. | On the other hand, a test is just intended to detect pitfalls or weaknesses in the system. |
| **Relationship with Other Life Cycle** | SDLC is taken as the predecessor | STLC is taken as the successor |
| **Team Involved** | Project Managers, Business analysts, Designers, and Developers, are involved in SDLC. | Quality assurance and Testers teams are involved in Software Testing Life Cycle (STLC). |
| **Prime goal** | The prime goal is to deliver a reliable and completely functional software product. | The prime goal is to check and confirm the software meets the particular requirements and functions appropriately. |
| **Distinct phases** | It comprises stages such as requirements gathering, design, execution, testing, delivery, and maintenance. | It comprises stages such as test planning, test design, test implementation, defect reporting & tracking, and test closure. |
| **Coverage** | It covers the complete software development process, from start to delivery/ deployment. | It covers the complete test process, starting from [test planning](https://www.browserstack.com/guide/test-planning) to test closure. |
| **Core Relationship** | SDLC is followed by STLC to validate and verify the software product. | STLC is an integral part of SDLC, ensuring the software is thoroughly tested before deployment. |
| **Outcome** | The ultimate outcome of SDLC is delivering a higher-quality product to the customer. | The ultimate outcome of STLC is preferably to deliver bug/ flaws-free software. |

**Q.30 what is the difference between Test cases, Test Scenarios, Test Script?**

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Test Case** | **Test Scenario** | **Test Script** |
| **Definition** | A detailed document with steps, inputs, and expected results | A high-level functionality to be tested | A set of automated instructions written in code |
| **Detail Level** | High – includes step-by-step instructions | Medium – just describes what to test | High – contains code to perform the test |
| **Purpose** | Validate a specific feature with defined inputs | Understand what functionality needs testing | Automate testing to save time and effort |
| **Used In** | Manual and Automation Testing | Test Planning and Design | Automation Testing Tools (e.g., Selenium) |
| **Example** | Steps to test login with valid credentials | Test login functionality | Python/Java code to automate login test |
| **Written By** | QA/Testers | QA/Testers/Test Leads | Automation Testers/Developers |
| **Format** | Document (Excel, Test Management Tool, etc.) | List or checklist format | Code (in scripting or programming language) |

**Q.31 Explain what Test plan is? What is the information that should be covered?**

A Test Plan is a formal document that outlines the strategy, scope, approach, resources, schedule, and activities required for testing a software application.

It acts as a roadmap for the entire testing process and ensures that everyone involved in the project is on the same page.

|  |  |
| --- | --- |
| **Section** | **Description** |
| ✅ **Test Plan ID** | Unique identifier for the test plan |
| 📋 **Introduction** | Brief about the project, features to be tested, and testing objectives |
| 🎯 **Test Objectives** | What you aim to achieve with the testing (e.g., validate login, check payments) |
| 🔍 **Scope of Testing** | Features to be tested and **not to be tested** (in-scope & out-of-scope) |
| 🛠️ **Test Approach/Strategy** | Manual or Automation? Types of testing: smoke, regression, functional, etc. |
| 📂 **Test Deliverables** | What you’ll produce: test cases, defect reports, test summary, etc. |
| 👥 **Team Roles & Responsibilities** | Who will do what (e.g., Test Lead, Testers, Developer, etc.) |
| 🗓️ **Schedule** | Timeline of when each testing activity will be performed |
| 🔧 **Test Environment** | Details about hardware/software setups needed for testing |
| 🧪 **Test Tools** | Any tools used: Selenium, JIRA, TestRail, etc. |
| ⚠️ **Risks and Mitigation** | Potential risks in the testing process and how to handle them |
| ✅ **Entry & Exit Criteria** | When testing can start (entry) and when it can end (exit) |
| 📊 **Metrics & Reporting** | How progress and quality will be tracked and reported |
| 📎 **Approvals** | Sign-offs from leads, managers, or clients |

**Example:**

If you're testing an e-commerce website, the test plan will say:

* What modules will be tested (Login, Cart, Payment)
* Who will test them
* What tools you'll use
* What documents will be delivered at the end
* When the testing will start and end

**TEST PLAN DOCUMENT**

**1. Test Plan ID**

TP\_EComm\_2025\_01

**2. Introduction**

This document outlines the test plan for the e-commerce web application. It includes the scope, objectives, resources, schedule, and deliverables for testing.

**3. Objectives**

* Verify login functionality
* Validate shopping cart and payment system
* Ensure responsiveness across devices

**4. Scope of Testing**

* **In-Scope:**

Login, Signup, Product Search, Cart, Checkout, Payment Gateway

* **Out-of-Scope:**

Admin panel, Backend database testing

**5. Test Strategy / Approach**

* Manual Testing
* Types of testing: Smoke, Functional, Integration, Regression
* Test cases will be created in Excel and tracked using JIRA

**6. Test Deliverables**

* Test Plan Document
* Test Cases
* Test Execution Report
* Defect Report
* Final Test Summary Report

**7. Roles and Responsibilities**

| **Role** | **Responsibility** | **Name** |
| --- | --- | --- |
| Test Lead | Planning and coordination | Amruta Jotkar |
| QA Engineer | Test case creation & execution | [QA Name] |
| Developer | Bug fixing | [Dev Name] |

**8. Schedule**

| **Activity** | **Start Date** | **End Date** |
| --- | --- | --- |
| Test Planning | 01-Apr-2025 | 03-Apr-2025 |
| Test Case Preparation | 04-Apr-2025 | 07-Apr-2025 |
| Test Execution | 08-Apr-2025 | 14-Apr-2025 |
| Defect Reporting & Retesting | 09-Apr-2025 | 15-Apr-2025 |

**9. Test Environment**

* Windows 10, Chrome/Firefox
* Mobile Devices (Android/iOS)
* Staging Server: test.ecommapp.com

**10. Test Tools**

* JIRA for bug tracking
* Excel for test case management
* Postman for API testing

**11. Risks and Mitigation**

| **Risk** | **Mitigation** |
| --- | --- |
| Delay in build delivery | Buffer time included in schedule |
| Unavailability of test environment | Use local setup temporarily |

**12. Entry and Exit Criteria**

* **Entry:**

All test cases created, test environment is ready, build is deployed

* **Exit:**

All critical test cases passed, no high-priority defects open

**13. Metrics & Reporting**

* Daily defect count
* Test case execution %
* Passed/Failed ratio

**14. Approvals**

| **Name** | **Role** | **Signature** |
| --- | --- | --- |
| Amruta Jotkar | Test Lead | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| [Manager Name] | Project Manager | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**Q.32 what is Priority?**

**Priority** refers to the **urgency** or **importance** of fixing a bug from the **business or customer perspective**.  
It is set by **Project Managers, Developers, or Business Analysts** depending on **how quickly** the defect should be fixed.

**Why is Priority Important?**

* Helps the development team **decide which bugs to fix first**.
* Ensures that issues affecting **core features, deadlines, or user experience** are resolved quickly.
* Helps in managing limited time and resources efficiently.

**Types of Priority:**

**Generally, there are 3 main levels (sometimes 4) of priority that testers and project managers use to classify bugs:**

1. **High Priority**

**Meaning**: The bug must be fixed **immediately**. It affects the **core functionality** or **critical business operations**.

Fixing it is urgent to continue the testing or release process.

Example:

* + Login is not working.
  + Payment gateway fails during checkout.
  + App crashes on startup.

**2. Medium Priority**

**Meaning**: The bug is important, but not **critical**. It should be fixed soon, but it doesn’t block the whole system.

Can be fixed in the next build or sprint.

**Example:**

* + Search filter gives incorrect results, but users can still search manually.
  + Button color is not as per design but still works fine.

**3. Low Priority**

* **Meaning:** The bug has very little impact on functionality or business.
* Can be fixed later, usually after major tasks are completed.
* **Example:**
  + Minor spelling mistake in Terms & Conditions.
  + Small design alignment issues.
  + Tooltip not showing when hovering over a button.

**4. Very Low / Cosmetic Priority (Optional)**

**Meaning**: Purely visual or cosmetic issue, doesn’t affect the user experience or functionality at all.

* Fix only if time allows or in future releases.
* **Example:**
  + Pixel difference in image alignment.
  + Extra space between two UI elements.

**Summary Table:**

| **Priority Type** | **Urgency** | **Impact Level** | **Fix Time** | **Example** |
| --- | --- | --- | --- | --- |
| **High** | Very Urgent | Blocks major features | Immediately | Login or payment failure |
| **Medium** | Urgent | Affects functionality | Soon | Filters not working |
| **Low** | Not urgent | Minor issue | Later | Typo, small UI bugs |
| **Very Low** | Optional | Cosmetic only | If time allows | Logo slightly off-center |

**Q.33 what is Severity?**

**Definition:**

**Severity** refers to **how serious or critical a defect (bug) is** in terms of **impact on the system's functionality**.

**In Simple Words:**

**Severity = Technical Impact** of the bug on the application.

It tells us how badly the software is **breaking** or **failing** due to the bug.

**Types of Severity:**

| **Severity Level** | **Description** | **Example** |
| --- | --- | --- |
| **Critical** | **The bug causes system crash, data loss, or blocks testing completely.** | **App crashes on login.** |
| **High** | **Major functionality is broken, but app is still running. No workaround.** | **Payment not processed correctly.** |
| **Medium** | **Some functionality is affected, but a workaround is available.** | **Search filter gives wrong results, but manual search works.** |
| **Low** | **Minor issue or cosmetic bug. Doesn’t affect main features.** | **Spelling mistake in help section.** |

**Severity vs Priority:**

| **Aspect** | **Severity** | **Priority** |
| --- | --- | --- |
| What it shows | **Technical seriousness** of the bug | **Business urgency** to fix it |
| Set by | Tester / QA | Test Lead / Project Manager |
| Focus | How badly it affects the system | How quickly it should be fixed |
| Example | Login fails = High Severity | If app launching soon = High Priority too |

**Example Scenarios:**

**Example 1:**

* Bug: App crashes when user clicks “Save”
* **Severity**: Critical
* **Priority**: High (if release is near)

**🔹 Example 2:**

* Bug: Spelling mistake in About Us page
* **Severity**: Low
* **Priority**: Low

**🔹 Example 3:**

* Bug: One report section gives wrong total
* **Severity**: High (wrong data!)
* **Priority**: Medium (if not used often)

**📝 Summary:**

* **Severity** = How **bad** the bug is (technical level)
* Set by **testers**
* Helps developers understand **how badly the system is affected**

**Q.34 what is Bug Life cycle?**

* **What is a Bug/ Defect?**

A **bug** or **defect** is an error, flaw, or unintended behavior in a software application that deviates from its expected outcome. Bugs can result from coding mistakes, unclear requirements, or unexpected use cases, impacting the functionality, performance, or user experience. Testers are responsible for finding and reporting defects to ensure the software works correctly and meets user needs. Fixing defects improves the overall quality of the product.

* **What is the Bug Life Cycle?**

The **Bug Life Cycle** is the standardized process a bug follows from identification to resolution, ensuring effective management and early detection to address issues promptly, preventing them from becoming deeply embedded in the code.

The bug life cycle in testing refers to a cycle of defects in which it goes through different states throughout its life.  The life cycle begins with a new defect discovered by a tester while testing the application. It continues until the tester discovers a specific solution and closes the bug, so it does not reoccur.

The overall [bug tracking](https://www.browserstack.com/guide/what-is-bug-tracking) life cycle involves multiple bug stages that enable the testers to track, debug, and improve the quality of the software.

### 10 Stages of a Bug Identification Workflow

**The diagram below depicts a bug throughout its lifecycle. Let’s go through ten stages of the workflow:**

**1. New:**This is the first stage in the life cycle of a bug. As a result, when a tester discovers a bug while testing applications, it falls into the ‘New’ category, and the bug is validated and tested in the subsequent stages of its life cycle.

**2. Assigned:**The bug is identified, approved by the testing lead, posted by the tester, and then assigned to the development team to work on. Finally, the testing team’s leader or [QA manager](https://www.browserstack.com/guide/top-skills-of-a-qa-manager) assigns the bug to the developer.

**3. Active/Open:**During this phase, the developer analyzes the bug and devises a solution. Suppose the developer doesn’t believe the bug requires any fixing. In that case, they can assign the bug to one of the four remaining stages: Duplicate Deferred, Rejected, or Not a Bug.

**4. Fixed:**After the developer analyzes the bug and makes the code changes to fix it, they can mark the bug as fixed and forward it to the testing team for further processing.

**5. Retest:**The tester retests the changed code, and the developer verifies with the testing team whether the specific bug has been fixed per the specified requirements.

**6. Closed:**This is the final stage of the bug life cycle. The tester retests it after the bug has been fixed. The tester changes the status from ‘Verified’ to ‘Close’ if they believe no further code is required and the bug has been successfully resolved. The closed stage shows that the bug is free of defects.

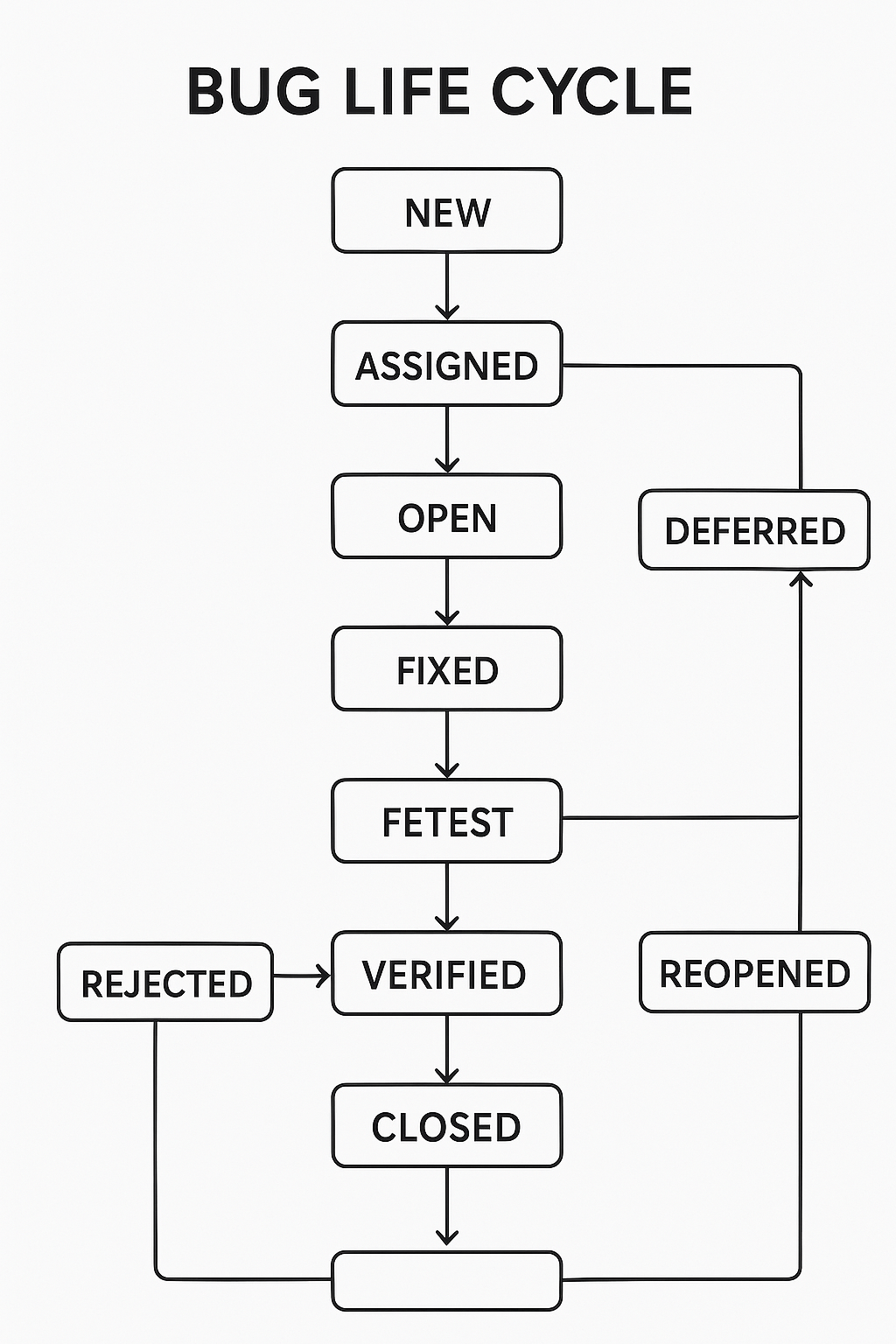
**7. Rejected:**The bug is generally rejected if the developer believes the bug is inaccurate. The bug’s status then changes to ‘Rejected.’

**8. Duplicate:**The developer marks the status as ‘Duplicate’ if the same bug occurs again or if the concept of the bug matches the concept of another same bug.

**9. Deferred:**When a bug is marked as deferred, it is of lower priority and can be fixed in the next release. The deferred stage comprises several bug-related events, such as low priority, less time to fix, or a bug that cannot cause a major issue with the software product.

**10. Not a bug:**The status of a specific bug is marked as ‘Not a Bug’ when the application product has little or no changes. The bug does not affect the program’s functionality, which limits its performance.

While going through the testing process to find out the bugs, the testers perform multiple tests like [usability](https://www.browserstack.com/guide/website-usability-testing), security, [performance](https://www.browserstack.com/guide/performance-testing), [accessibility](https://www.browserstack.com/guide/accessibility-testing), interface, and [functionality](https://www.browserstack.com/guide/functional-testing).

But the tests are incomplete without the compatibility test or the [cross-browser testing](https://www.browserstack.com/cross-browser-testing#:~:text=Cross%20browser%20testing%20helps%20with,work%20on%20their%20browser%2DOS.). 

**Q.35 Bug categories are?**

Here are the most common types of software bugs or defects encountered in software testing so that developers and testers can deal with them better.

#### ****1. Functional Bugs****

Functional bugs are associated with the functionality of a specific software component.

In simple terms, any component in an app or website that doesn’t function as intended is a functional bug.

Such bugs are often detected when testers conduct comprehensive [functional testing](https://www.browserstack.com/guide/functional-testing) for their apps or websites in [real user conditions](https://www.browserstack.com/real-user-conditions-testing-on-browserstack). Teams need to ensure that all the functional bugs are resolved in the early stages so as to avoid delivering bad user experiences in the production environment.

For example, a Login button doesn’t allow users to login, an Add to cart button that doesn’t update the cart, a search box not responding to a user’s query, etc.

#### ****2. Logical Bugs****

A logical bug disrupts the intended workflow of software and causes it to behave incorrectly. These bugs can result in unexpected software behavior and even sudden crashes. Logical bugs primarily take place due to poorly written code or misinterpretation of business logic.

For example of logical bugs include:

* Assigning a value to the wrong variable.
* Dividing two numbers instead of adding them together resulting in unexpected output

#### ****3. Workflow Bugs****

Workflow bugs are associated with the user journey (navigation) of a software application.  
Let’s consider an example of a website where a user needs to fill up a form regarding their medical history. After filling the form, the user has three options to choose from:

* Save
* Save and Exit
* Previous Page

From the available options, if the user clicks on “Save and Exit,” the user intends to save the entered information and then exit. However, if clicking on the Save and Exit button leads to an exit from the form without saving the information, it leads to a workflow bug.

#### ****4. Unit Level Bugs****

Unit level bugs are very common, and they are typically easier to fix. Once the initial modules of software components are developed, developers perform unit testing to ensure that the small batches of code are functioning as expected. Here’s where developers encounter various bugs that get overlooked in the coding stages. Unit level bugs are easier to isolate as developers deal with a comparatively small amount of code. Moreover, replicating these bugs takes less time, so developers can track the exact bug and fix it in no time.

For example, if a developer creates a single page form, a unit test will verify whether all the input fields are accepting appropriate inputs and validate buttons for functionality. In case a field doesn’t accept the appropriate characters or numbers, developers encounter a unit-level bug.

#### ****5. System-Level Integration Bugs****

System-level integration bugs primarily pop up when two or more units of code written by different developers fail to interact with each other. These bugs primarily occur due to inconsistencies or incompatibility between two or more components. Such bugs are difficult to track and fix as developers need to examine a larger chunk of code. They are also time-consuming to replicate. Memory overflow issues and inappropriate interfacing between the application UI and the database are common examples of system-level integration bugs.

For example: An online booking system integrates with multiple third-party service providers (e.g., airlines, hotels). If one of the service providers experiences high latency or timeouts, the entire booking process may fail, resulting in incomplete bookings or incorrect availability information.

#### ****6. Out of Bound Bugs:****

Out of Bound Bugs show up when the system user interacts with the UI in an unintended manner. These bugs occur when an end-user enters a value or a parameter outside the limits of unintended use.

For example, entering a significantly larger or a smaller number or entering an input value of an undefined data type. These bugs often pop up in form validations during functional testing of web or mobile apps.

**7. Security Bugs**

Security is a major concern for software development. Security Bugs are a major risk for users and should be taken very seriously and resolved. Due to their high severity and vulnerable nature, security bugs are considered among the most sensitive bugs of all types and should be handled with criticality and urgency.

These bugs might not hinder the operation but can compromise the whole system. These should be checked thoroughly at regular intervals.

A common example is SQL injection, where an attacker can manipulate a database query to gain unauthorized access.

**8. Performance Bugs**

Performance bugs occur when a software application fails to meet the expected performance benchmarks, such as load times, response times, or throughput. These bugs can significantly degrade the user experience, especially in high-traffic or resource-intensive environments.

For example: An e-commerce website experiences a performance bug where the page load time exceeds 5 seconds during peak traffic hours, causing frustration for users and leading to a high abandonment rate.

**9. Compatibility Bugs**

Compatibility bugs arise when a software application does not function correctly across different environments, devices, or platforms. These bugs can lead to inconsistent user experiences and reduced accessibility.

For example: A mobile app works perfectly on Android devices but crashes or displays incorrectly on certain iOS devices, leading to a compatibility bug that impacts a significant portion of the user base.

**10. Usability Bugs**

Usability bugs affect the overall user experience, making it difficult or confusing for users to interact with the software. These bugs do not necessarily prevent functionality but can lead to poor user satisfaction and increased user error rates.

For example: A web application has a complex navigation structure that makes it difficult for users to find essential features, leading to a usability bug that frustrates users and reduces engagement.

**11. Concurrency Bugs**

Concurrency bugs occur in software systems that involve parallel processing or multi-threading. These bugs arise when multiple threads or processes interact in unintended ways, leading to unpredictable behavior, data corruption, or system crashes.

For example: A banking application experiences a concurrency bug where two users attempt to transfer funds simultaneously, leading to incorrect account balances or duplicate transactions.

**Q.36 Advantages Bugzilla?**

**Advantages of Bugzilla**

1. **Open Source & Free**
   * No license cost; freely available and customizable.
2. **Customizable Workflow**
   * You can define your own bug lifecycle, user roles, and permissions.
3. **Advanced Search & Filters**
   * Powerful search options with saved filters and email notifications.
4. **Bug Tracking & Management**
   * Tracks bugs efficiently with features like severity, priority, component, etc.
5. **Email Notifications**
   * Sends automatic email updates for bug changes, assignments, and status updates.
6. **Reporting & Charts**
   * Provides bug statistics, graphs, and reports for better analysis.
7. **Web-Based Interface**
   * Easy access from any device or platform through a web browser.
8. **Security**
   * Granular access control with user roles and permissions.
9. **Attachment Support**
   * Allows uploading screenshots, logs, or documents with bug reports.

10. **Integration**

* Can integrate with version control systems like Git and other tools like Testopia.

**Q.37 What are the different methodologies in Agile development model?**

**Below is a detailed explanation of what these Agile Methodologies are.**

**1. Scrum**

It focuses primarily on how to handle tasks in a team-based development setting, and it is an agile development methodology. Scrum basically evolved from activities that take place during a rugby round. Scrum promotes operating in small teams and thinks that the development team should be empowered (say- 7 to 9 members). Three roles make up Agile and Scrum, and their duties are described below:

* **Scrum Lead**: The scrum master is in charge of organizing the team, the sprint meeting, and removing roadblocks.
* **Product creator**: The product owner builds the product backlog, organizes it by priority, and is in charge of delivering features at each iteration.
* **Agile Team**: Team coordinates and oversees its own work to finish the sprint or cycle.

**Product Backlog in Scrum**

The number of requirements (user stories) that need to be finished for each release is tracked in this repository. The Product Owner should keep track of it, prioritize it, and share it with the scrum team. The team may also ask for the addition, modification, or elimination of a new requirement.

**Scrum Practices**

Scrum methodologies’ workflow:

* A sprint is one iteration of a scrum.
* The list of information needed to produce the final product is called the product backlog.
* The most important user stories from the Product backlog are chosen and added to the Sprint backlog.
* Working together on the specified sprint backlog.
* Team verifies everyday work for accuracy.
* The team provides product functionality at the end of the sprint.

**2. Extreme Programming (XP)**

The Agile framework for software development processes most closely resembles XP. It strives to build high-quality software while also simplifying the entire process for the development team. XP places a high importance on feedback, communication, simplicity, bravery, and respect.

It works best when,

* The criteria are always shifting.
* Team deadlines are constrained.
* Stakeholders desire to lower risk while meeting timelines.
* Unit and functional testing can be automated by teams.

**3. Adaptive Software Development (ASD)**

In the first decade of the 1990s, Sam Bayer and Jim Highsmith built adaptive software development (ASD). It comprises the notions of continual adaptation, or how to adopt change rather than avoid it. Learn, collaborate, and speculate is the term of the dynamic growth process that is utilized in ASD. Because the business setting is continuously changing, this process is concentrated on close customer and development engagement and ongoing learning.

ASD offers a non-linear iterative life cycle, in contrast to most software development methodologies, which use a static life cycle, i.e., Plan-Design-Build. Each process can iterate and be altered while another process is being carried out. It suggests rapid application development, which places an emphasis on speed of development to produce a high-quality, low-maintenance product that involves the user as much as possible. The following are the primary traits of ASD:

Speculate:

The major objectives and aims of the project must be established during this phase of its beginning by comprehending the constraints (risk areas) within which it must work. Maintaining coordination between teams during this phase ensures that what is learned by one team is communicated to the others and does not need to be acquired again by other groups from scratch. This phase is where the majority of the development is concentrated.

Learn:

The last stage involves several collaboration cycles, and the goal is to record all of the lessons learnt, both good and bad. The prosperity of the project relies on this stage.

**4. Dynamic Software Development Method (DSDM)**

A group of specialists and vendors in the sector of software development built the Dynamic Software Development Method in 1994. Software programs with budgets and constrained schedules are the primary priority of DSDM. It accentuates regular product process delivery, and growth is iterative and gradual.

With the Dynamic Software Development Method (DSDM), a roadmap of continuous and early deliveries can be created for the project. This allows for the implementation of an incremental solution, adaptation in response to feedback received along the way, and verification that the anticipated benefits are being realized.

The DSDM is an agile model that may unquestionably assist companies used to operating on projects to alter their mindset and method of operation in order to increase their ability to create value and shorten time to market.

**5. Feature Driven Development (FDD)**

The key component of this methodology is “designing & creating” features. FDD, in contrast to other Agile development techniques in software engineering, outlines very precise and condensed work phases that must be completed separately for each feature. Domain walkthrough, design review, promotion to build, code review, and design are all included. FDD creates products with the target market in mind.

* Visibility of progress and results
* Regular Builds
* Configuration Management
* Inspections
* Feature Teams
* Component/ Class Ownership
* Development by feature
* Domain object Modeling

**6. Kanban**

Without adding to the stress of the software development lifecycle, Kanban is a highly visual workflow management technique that enables teams to actively supervise product creation, with a focus on continuous delivery (SDLC). It has gained popularity among groups that use Lean software development techniques.

The three fundamental tenets of Kanban are to visualize the workflow, reduce the amount of work that is in process, and enhance the flow of work. The Kanban technique is intended to aid teams in collaborating more effectively, much like Scrum is. It promotes an atmosphere of active and continuing learning and growth by encouraging continuous collaboration and attempting to establish the ideal process.

**7. Behavior Driven Development (BDD)**

A behavior-focused agile system development methodology is called behavior driven development (BDD). It was created by Dan North in 2003 as an extension of the TDD methodology. Dan North tried to include non-technical people when creating the system’s technological functionality. Inadvertently leaving out business principles that are already part of the functionality when developing software can occasionally lead to repeated and even serious defects.

BDD uses universal language concepts to facilitate communication inside a software project between persons with and without technical expertise. The BDD development process is built on the writing of test cases and features. These provide the guidelines and requirements for proper system operation. It defines what is necessary for the functionality to start, what will happen next, and what the results will be after it has been completed. Teams who use BDD are better able to communicate needs clearly, find bugs quickly, and create long-lasting software.



**Q.38 When to use usability Testing?**

### ****When to Use Usability Testing****

#### 1. ****Before Launching a Product****

* To ensure the app or website is **intuitive** and **easy to navigate**.
* Identify major **user pain points** before going live.

#### 2. ****During UI/UX Design Phase****

* To test **wireframes, prototypes**, or design mockups.
* Helps designers make better decisions based on **real user feedback**.

#### 3. ****When Introducing a New Feature****

* To make sure the new feature is **understood and usable** by users.
* Prevents confusion and reduces support requests.

#### 4. ****When User Complaints or Drop-offs Increase****

* If users are leaving the site or app without completing tasks.
* Helps find **usability flaws** causing frustration.

#### 5. ****Before a Major Redesign****

* Compare old and new versions to see which one performs better.
* Ensures the new design actually improves **user experience**.

#### 6. ****In Accessibility Testing****

* Ensure that users with **different abilities** (e.g., visual, hearing impairments) can use your product comfortably.

#### 7. ****When You Want to Increase Conversions****

* Usability issues often hurt **sales, signups, or task completions**.
* Testing helps improve those metrics by making the flow smoother.

### Real-life Example:

You build a shopping website. During usability testing, users struggle to find the “Add to Cart” button because it’s hidden. That insight helps you fix the UI before it affects real sales.

**Q.39 What is the procedure of GUI Testing?**

GUI (Graphical User Interface) Testing is a process used to ensure that the visual elements of a software application function correctly and meet design specifications. The procedure involves verifying elements like buttons, menus, icons, and overall user experience. Here's a standard step-by-step **procedure of GUI Testing**:

**1. Test Planning**

* **Define scope**: Determine what parts of the GUI need to be tested (e.g., forms, buttons, navigation).
* **Create test strategy**: Choose between manual testing, automated tools (like Selenium, TestComplete), or a combination.
* **Select tools**: Based on your application type (web, mobile, desktop), select appropriate testing tools.

**2. Test Case Design**

* **List GUI elements**: Identify all components like buttons, text fields, labels, drop-downs, etc.
* **Create test cases** for:
  + Element visibility and alignment
  + Functionality (e.g., clicking a button triggers the expected action)
  + Input field validation
  + Navigation and workflow
  + Consistency with design specs (e.g., color, font)
  + Responsiveness (screen resolution compatibility)

**3. Test Environment Setup**

* Configure test environment:
  + Right OS, browser versions, and screen sizes
  + Ensure access to UI elements
  + Set up automated testing framework if needed

**4. Test Execution**

* **Manual Testing**: Check each element for expected behavior manually.
* **Automated Testing**: Use scripts to run repetitive checks (e.g., UI regression tests).
* **Check for**:
  + Broken or misaligned elements
  + Incorrect labels or text
  + Unresponsive components
  + Workflow or navigation issues

**5. Defect Reporting**

* Log UI bugs found during testing.
* Include screenshots, steps to reproduce, and expected vs actual behavior.
* Assign severity and priority for fixes.

**6. Regression Testing**

* After bugs are fixed, re-run GUI tests to ensure that changes haven't affected other parts of the UI.

**7. Final Verification**

* Conduct end-to-end GUI walkthrough.
* Make sure the GUI aligns with the final design documents.
* Perform cross-platform and cross-browser validation if applicable.

**Tools Commonly Used**

* **Selenium** (for web)
* **Appium** (for mobile)
* **TestComplete**, **Ranorex**, **Katalon Studio**
* **BrowserStack** or **Sauce Labs** for cross-browser testing

**Q.40 Explain the difference between Authorization and Authentication in web testing. What are the common problems faced in web testing?**

The difference between Authorization and Authentication in web testing, followed by a list of common problems faced in web testing:

### 

### Authentication vs. Authorization

| **Feature** | **Authentication** | **Authorization** |
| --- | --- | --- |
| **Definition** | Verifying the **identity** of a user (Are you who you say you are?) | Determining what **permissions or access level** a user has |
| **Focus** | Identity check (username, password, OTP, biometrics) | Access control (can the user access this page/data?) |
| **Occurs When?** | First step before giving access to the system | After authentication is successful |
| **Example** | Logging in with email and password | Allowing only admins to delete records |
| **Test Cases Include** | - Valid/invalid login |  |

* Forgotten password
* Multi-factor auth  
  | - Access control for different roles
* Forbidden pages
* URL access restrictions |