Unit 4
Software Testing

What is Software Testing?

Software testing is an important process in the **Software Development Lifecycle(SDLC)**. It involves **verifying** and **validating** that a **Software Application** is free of bugs, meets the technical requirements set by its Design and Development, and satisfies user requirements efficiently and effectively.

Software Testing is a process of verifying and validating whether the **Software Product** or **Application** is working as expected or not. The complete testing includes identifying errors and bugs that cause future problems for the performance of an application.

Objective of software Testing

Software Testing is a critical quality assurance process in the Software Development lifecycle. Its core objectives extend beyond mere error identification; they encapsulate a range of essential goals that collectively enhance the overall software quality.

a) Identifying defects and errors: *One of the primary Objectives of Software Testing is to unearth defects and mistakes within the software code and functionality. By meticulously probing the application, Testers ensure that potential issues are identified, documented, and rectified before the software is released.*

b) Verifying correctness: *Testing verifies that the software behaves as intended and meets its functional requirements. It aims to confirm that each feature, module, and component functions accurately and consistently, aligning with the specifications.*

c) Validating user requirements: *Ensuring the software meets user expectations is another key objective. Through testing, developers verify that the software aligns with user requirements, delivering a product that caters to user needs.*

d) Ensuring reliability and performance: Software Testing aims to guarantee the application's reliability, stability, and performance under various conditions. It assesses how the software functions in real-world scenarios and under different loads to ensure a consistent and efficient user experience.

Types of Manual Testing

1. **White Box Testing**
   White Box Testing is a software testing technique that involves testing the internal structure and workings of a software application. 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.
2. **Black Box Testing**
   Black-Box Testing is a type of software testing in which the tester is not concerned with the internal knowledge or implementation details of the software but rather focuses on validating the functionality based on the provided specifications or requirements.
3. **Gray Box Testing**
   Gray Box Testing is a software testing technique that is a combination of the **Black Box Testing** technique and the **White Box Testing** technique. In the Black Box Testing technique, the tester is unaware of the internal structure of the item being tested and in White Box Testing the internal structure is known to the tester.
4. **Unit Testing**
   Unit Testing is a method of testing individual units or components of a software application. It is typically done by developers and is used to ensure that the individual units of the software are working as intended.
5. **Integration Testing**
   Integration Testing is a method of testing how different units or components of a software application interact with each other. It is used to identify and resolve any issues that may arise when different units of the software are combined.
6. **System Testing**
   System Testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users
7. **Acceptance Testing:** This is a kind of testing conducted to ensure that the requirements of the users are fulfilled before its delivery and that the software works correctly in the user's working environment.
8. **Regression Testing** is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. We can also say it is nothing but a full or partial selection of already executed test cases that are re-executed to ensure existing functionalities work fine.This type of testing is done to ensure that new code changes do not have any side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

Difference between Performance testing and Functional testing:

Aspect	Functional Testing	Performance Testing
Purpose	Verifies that software functions as intended and meets specified requirements.	Evaluates the system's performance under various conditions like load, stress, and scalability.
Focus	Tests individual functions or features to ensure correct behavior.	Measures responsiveness, speed, and stability of the entire system.
Scope	Includes unit testing, integration testing, system testing, etc.	Involves load testing, stress testing, scalability testing, etc.
Testing Criteria	Validates functionality, user interface, data handling, etc.	Assesses speed, reliability, scalability, and resource usage.
Examples	Unit testing, integration testing, system testing, acceptance testing, etc.	Load testing, stress testing, endurance testing, scalability testing, etc.
Key Metrics	Pass/fail based on functional requirements.	Response time, throughput, resource utilization, error rates, etc.
Users' Perspective	Concerned with what the system does.	Concerned with how well the system performs under different conditions.
Tools	Selenium, JUnit, TestNG, etc.	Apache JMeter, LoadRunner, Gatling, etc.

Difference between Stubs and Drivers

In software testing, **stubs** and **drivers** are essential components used to simulate the behavior of missing or incomplete modules. They help in testing the functionality of a system even when some parts are not yet developed.

Stubs

Stubs are dummy implementations of modules or functions that are called by the component being tested. They simulate the behavior of the dependent modules that are not yet developed or available for testing. Stubs are typically used in **Top-Down Integration Testing**.

Key Features of Stubs:

- **Definition**: Stubs are placeholder implementations or simulated modules used in place of actual modules or components that a module being tested depends on.
- **Purpose**: Stubs are used when a module being tested relies on the functionality of another module, which may not be available or fully implemented at the time of testing.
- **Functionality**: Stubs provide basic functionality or predetermined responses that allow the module being tested to proceed with its execution.
- **Dependency**: Stubs represent the dependent modules that the module being tested relies on.
- **Testing Focus**: Stubs are primarily used in top-down testing approaches, where higher-level modules are tested before lower-level modules.

Example:

In a client-server architecture, if the client module is being tested, a stub can be used to simulate the server's functionality and respond to the client's requests[3].

Drivers

Drivers are dummy implementations used to simulate the behavior of higher-level modules or components that invoke the component being tested. Drivers are employed in **Bottom-Up Integration Testing**[1][2].

Key Features of Drivers:

- **Definition**: Drivers are software components that enable the testing of a module in isolation by simulating the behavior and functionality of the higher-level modules that interact with it[1].
- **Purpose**: Drivers are used when a module being tested requires input or interaction from other modules that are not yet developed or available[2].
- **Functionality**: Drivers provide the necessary input or interaction to the module being tested to simulate the behavior of the higher-level modules[3].
- **Dependency**: Drivers represent the calling modules that interact with the module being tested[1].

- **Testing Focus**: Drivers are commonly used in bottom-up testing approaches, where lower-level modules are tested before higher-level modules[2].

Example:

In a software system with a layered architecture, if the core processing module is being tested, a driver can be used to simulate the behavior of the user interface module and provide input to the core processing module[3].

Conclusion

Both **stubs** and **drivers** are crucial in facilitating the testing and integration of software components. Stubs act as placeholders for dependent modules that are not fully implemented, allowing the module being tested to proceed. On the other hand, drivers simulate the behavior of higher-level modules to provide input or interaction for the module being tested[12]. They are essential tools in ensuring the correctness and efficiency of the software testing process.

White Box Testing

White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

The term "WhiteBox" was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "Black Box Testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

What Does White Box Testing Focus On?

White-box Testing focuses on the internal workings of an application, ensuring that its logic, structure, and flow operate as intended. Unlike black-box testing, which focuses on user interactions without knowledge of the underlying code, white-box testing involves examining the software's source code directly. Below are the key areas of focus in white-box testing:

1. Code Logic and Flow

Checks if the program's logic works as intended. This means verifying that code modules (like functions or classes) interact correctly and that control structures such as if-else statements, loops, or switches execute properly. For example, ensuring a login function redirects users correctly based on valid or invalid credentials.

2. Code Coverage

Ensures tests exercise as much of the code as possible. This includes:

- **Statement coverage**: Every line of code runs at least once.
- **Branch coverage**: All decision paths (e.g., true/false conditions) are tested.
- **Path coverage**: Every possible route through the code is checked.
  This helps find untested or "dead" code that could hide bugs.

3. Data Flow and Variables

Verifies that data is passed and manipulated correctly through the application. This includes ensuring variables are properly initialized, updated, and used without causing any errors or unexpected behavior. Monitoring the flow of data ensures that the system remains stable and reliable as it processes inputs, calculations, and outputs.

4. Internal Functions and Methods

Tests the individual functions or methods to ensure they perform their intended tasks accurately and return the expected results. This part of white-box testing focuses on validating business logic, mathematical computations, and other operations within the software. Ensuring these internal processes are correct helps catch potential issues early and improves the reliability of the overall system.

5. Boundary Conditions

Examines how the code handles edge cases, like the maximum or minimum values for inputs (e.g., a loop running 0 or 100 times, or an input field accepting a 255-character string). This ensures the app doesn't crash at its limits.

6. Error Handling and Exception Management

Confirms the program manages errors smoothly, catching exceptions (e.g., invalid inputs) and providing clear feedback. For example, testing if a file upload function handles a missing file gracefully.

Types of White Box Testing

White box testing can be done for different purposes at different places. There are three main types of White Box testing which is follows:-

1. **Path Testing**: White box testing will be checks all possible execution paths in the program to sure about the each one of the function behaves as expected. It helps verify that all logical conditions in the code are functioning correctly and efficiently with as properly manner, avoiding unnecessary steps with better code reusability.
2. **Loop testing**: It will be check that loops (for or while loops) in the program operate correctly and efficiently. It checks that the loop handles variables correctly and doesn't cause errors like infinite loops or logic flaws.
3. **Unit Testing**: Unit Testing checks if each part or function of the application works correctly. It will check the application meets design requirements during development.

4. **Mutation Testing**: It is a type of Software Testing that is performed to design new software tests and also evaluate the quality of already existing software tests. Mutation testing is related to modification a program in small ways.
5. **Integration Testing**: Integration Testing Examines how different parts of the application work together. After unit testing to make sure components work well both alone and together.
6. **Penetration testing**: Penetration testing, or pen testing, is like a practice cyber attack conducted on your computer systems to find and fix any weak spots before real attackers can exploit them. It focuses on web application security, where testers try to breach parts like APIs and servers to uncover vulnerabilities such as code injection risks from unfiltered inputs.

Black Box Testing

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

Types of Black Box Testing

The testing of application without knowing the internal code or structure, following are the various Types of Black Box Testing:

1. Functional Testing

Functional Testing is a type of Software Testing in which the system is tested against the functional requirements and specifications. Functional testing ensures that the requirements or specifications are properly satisfied by the application.

- This testing is not concerned with the source code of the application. Each functionality of the software application is tested by providing appropriate test input, expecting the output, and comparing the actual output with the expected output.
- This testing focuses on checking the user interface, APIs, database, security, client or server application, and functionality of the Application Under Test. Functional testing can be manual or automated. It determines the system's software functional requirements.

2. Regression Testing

Regression Testing is like a Software Quality checkup after any changes are made. It involves running tests to make sure that everything still works as it should, even after updates or tweaks to the code. This ensures that the software remains reliable and functions properly, maintaining its integrity throughout its development lifecycle.

- Regression means the return of something and in the software field, it refers to the return of a bug. It ensures that the newly added code is compatible with the existing code.

- In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.

3. Nonfunctional Testing

Non-functional Testing is a type of Software Testing that is performed to verify the non-functional requirements of the application. It verifies whether the behavior of the system is as per the requirement or not. It tests all the aspects that are not tested in functional testing.

- It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

- It is as important as functional testing.

- It is also known as NFT. This testing is not functional testing of software. It focuses on the software's performance, usability, and scalability.

Advantages of Black Box Testing

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used to find the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing

- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.

- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

Difference between Black Box Testing and White Box Testing

Parameters	Black Box Testing	White Box Testing
Definition	Black Box Testing is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	White Box Testing is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
Testing objectives	Black box testing is mainly focused on testing the functionality of the software, ensuring that it meets the requirements and specifications.	White box testing is mainly focused on ensuring that the internal code of the software is correct and efficient.
Testing methods	Black box testing uses methods like equivalence partitioning, boundary value analysis, and error guessing to create test cases.	White box testing uses methods like control flow testing, data flow testing and statement coverage testing.
Knowledge level	Black box testing does not require any knowledge of the internal workings of the software, and can be performed by testers who are not familiar with programming languages.	White box testing requires knowledge of programming languages, software architecture and design patterns.
Scope	Black box testing is generally used for testing the software at the functional level.	White box testing is used for testing the software at the unit level, integration level and system level.

What is a Software Test Suite?

A test suite is a methodical arrangement of test cases that are developed to validate specific functionalities. Individual test cases in a suite are created to verify a particular functionality or performance goal. All the test cases in a test suite ultimately used to verify the quality, and dependability of a software.

What is Software Test Suite Composed of?

A test suite is composed of the items listed below −

- **Test Cases** − They describe the particular input situations, steps to execute, expected results to test a specific feature of the software.
- **Test Scripts** − They describe a group of automated sequences of commands that are needed to execute a test case. They can be developed using multiple languages, and are utilized to automate the testing activities.
- **Test Data** − They constitute the set of inputs required at the time of test execution. They play a very important role in verifying multiple scenarios, and situations.

Types of Software Test Suites

The different types of software test suites are listed below −

1. Functional Test Suites

They are used to verify if a particular functionality in the software is working as expected. For example, the payment functionality in a software.

2. Regression Test Suites

They are re-executed each time there are new code changes. They verify if those changes have not impacted the existing functionalities of the software. For example, the regression test suite is executed at the end each sprint.

3. Smoke Test Suites

They are executed to verify the basic functionalities on a new build of the software and to confirm whether the same build can be used for further testing.

4. Integration Test Suites

They are executed to verify the communications between various modules after they have been integrated. For example, the changes made in the front end of the software should reflect in the backend as well.

Steps to Create a Software Test Suite

The steps to create a software test suite are listed below −

Step 1 − The first step is to identify the aims and objectives of testing. It also includes verification of the features, performance parameters, and integrations.

Step 2 − The second step is to select and create the test cases as per the test objectives identified in the step1. The test cases should contain details such as test steps, data, and the expected outcomes. Each test should be independent, unrelated, reusable, and easily maintainable.

Step 3 − The third step is to segregate the test cases as logical units and to prioritize them as per their criticality, functionalities, and execution sequence. The dependencies and preconditions for every test case are explicitly defined.

Step 4 − The fourth step is to select the appropriate automation tools, and frameworks to generate and manage the test scripts.

Step 5 − The fifth step is to develop the test scripts using the chosen tool, if automation testing is adopted. The test environment is configured with the required resources and test data. The test suite is verified to check if it is correctly developed, and can be triggered at any time for execution.

How to Execute a Software Test Suite?

The steps to execute a software test suite are listed below −

Step 1 − The first step is to configure the test environment (which is a copy of the production environment) along with the required resources, test data, and dependencies.

Step 2 − The second step is to decide the order of test execution to make the test suite more productive, and to catch important defects earlier. It also includes considering the dependencies between the test cases, and prioritizing them accordingly.

Step 3 − The third step is to trigger the test suite for execution using the automation tools. For manual execution, the testers execute the documented steps one by one, and record the outcomes correctly.

Step 4 − The fourth step is to keep track of the complete execution process, and to determine any bottlenecks. The results of each test case are logged along with their outcomes, error messages, and other important information.

Step 5 − The fifth step is to evaluate the failed test cases to determine the cause of failures, to identify the defects in the software and to detect the environmental issues.

Step 6 − The fifth step is to prepare the test result, and share it to the project stakeholders.

Step 7 − The seventh step is to retest the fixed bugs, and to re-trigger execution for the corresponding parts of the test suite to ensure that all the issues have been fixed.

What is Alpha Testing?

Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance tests. It is the first stage of software testing, during which the internal development team tests the program before making it available to clients or people outside the company.

Key points to remember:

Following are the key points related to Alpha Testing:

- **Work Done by Developers**: The internal development team, which consists of developers and testers, usually conducts alpha testing in a controlled setting.
- **Goal:** Finding and fixing bugs, flaws, and usability issues is the main goal before releasing the product for external users or wider testing.
- **Little User Engagement:** Alpha testers are few in number and frequently comprise members of the development team or those intimately connected to the project.
- **Environment:** Alpha testing is typically carried out in a development environment or laboratory that resembles actual settings.

What is Beta Testing?

Beta Testing is performed by real users of the software application in a real environment. Beta testing is one type of User Acceptance Testing. A pre-release version of the product is made available for testing to a chosen set of external users or customers during the second phase of software testing.

Key Points to remember:

Following are the key points related to Beta Testing:

- **Actions Taken by Users:** Customers or other users outside the development team participate in beta testing. The software is available to these users prior to its official release.
- **Goal:** The primary objective is to get input from actual users in order to find any bugs, usability difficulties, or areas that need to be improved before the product is formally released.
- **Greater User Participation:** In order to capture a variety of viewpoints, a larger number of users—including a varied range, can serve as beta testers.
- **Environment:** Real-world settings are used for beta testing to simulate how users will interact with the program while performing daily duties.

Difference between Alpha and Beta Testing:

The difference between Alpha and Beta Testing is as follows:

Parameters	**Alpha Testing**	**Beta Testing**
Technique Used	Alpha testing uses both white box and black box testing.	Beta testing commonly uses black-box testing.

Parameters	Alpha Testing	Beta Testing
Performed by	Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Performed at	Alpha testing is performed at the developer's site.	Beta testing is performed at the end-user of the product.
Reliability and Security	Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Ensures	Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Requirement	Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Execution	Alpha testing may require a long execution cycle.	Beta testing requires only a few weeks of execution.
Issues	Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from the beta testing will be implemented in future versions of the product.
Test Cycles	Multiple test cycles are organized in alpha testing.	Only one or two test cycles are there in beta testing.

What is Static Testing?

Static Testing also known as Verification testing or Non-execution testing is a type of Software Testing method that is performed in the early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that cannot be found using Dynamic Testing, can be easily found by Static Testing.

1. Static can be done manually or with the help of tools to find bugs and improve the quality of the software.
2. It helps to find errors in the early stage of development which is also called the verification process.
3. It enhances maintainability and ultimately saves time and money in the long run.

Need for Static Testing

Static testing is needed whenever the following situations are encountered while testing an application or software:

1. **Increased software size:** Static testing is required to get free from bugs in the early stages of development as with testing activity the size of the software increases which is difficult to handle due to a reduction in the productivity of the code coverage.
2. **Dynamic testing is expensive:** Dynamic testing is more expensive than static testing as dynamic testing uses test cases that have been created in the initial stages and there is also a need to preserve the implementation and validation of the test cases which takes a lot of time from test engineers:
3. **Dynamic testing is time-consuming:** Static testing is required as dynamic testing is a time-consuming process.
4. **Bugs detection at early stages:** Static testing is helpful as it finds bugs at early stages, while dynamic testing finds bugs at later stages which makes it time-consuming and costly to fix the bugs.
5. **Improvement of development productivity:** Static testing helps to identify bugs early in the software development thus it helps to reduce the flaws during production and increase development productivity.

Objectives of Static Testing

Below are some of the objectives of static testing:

1. **Decreases Flaws:** Static testing will decrease the flaws in production as the bugs will be detected early in the software development.
2. **Saves Time:** Early detection of the bugs helps to save a lot of time, effort, and cost that will be required to fix the bugs.
3. **Easy Bug Fixing:** Static testing is used to identify the bugs early in software development, where it is quite easy to fix the bugs.

4. **Quality Improvement:** Enhance overall software quality of the software product by ensuring the compliance with the coding standards and best practices.
5. **Cost Efficiency:** Static testing helps to reduce the cost associated with dynamic testing by catching defects early.

Features Tested in Static Testing

Static testing involves testing the following things:

1. Unit Test Cases: It ensures test cases are complete, written in the correct manner, and follow the specified standards.
2. Business Requirements Document (BRD): It verifies that all business requirements are clearly mentioned in the documentation.
3. Use Cases: It examines the use cases so that they accurately represent user interactions with the system.
4. Prototype: It reviews the prototype to make sure that it accurately represents the main design and functionality.
5. System Requirements: It check the system requirements document for complete accuracy.
6. Test Data: It reviews the test data to ensure it is complete and covers all possible input scenarios.
7. Traceability Matrix Document: This ensures that all requirements are mapped to corresponding test cases.
8. Training Guides: It reviews training materials to make sure that they accurately reflect the system functionality and user procedures.
9. Performance Test Scripts: It examines performance test scripts to ensure they cover all critical performance aspects.

Benefits of Static Testing

Below are some of the benefits of static testing:

1. **Early defect detection:** Static testing helps in early defect detection when they are most easy and cost-effective to fix.
2. **Prevention of common issues:** Static testing helps to fix common issues like syntax errors, null pointer exceptions, etc. Addressing these issues early in development helps the teams to avoid problems later.
3. **Improved code quality:** Static testing helps to make sure that the code is easy to maintain and well-structured. This leads to a higher quality code.
4. **Reduced costs:** Early bug detection in static testing helps to fix them early in the development thus saving time, effort, and cost.

5. **Immediate feedback:** Static testing provides immediate evaluation and feedback on the software during each phase while developing the software product.
6. **Helps to find exact bug location:** Static testing helps to find the exact bug location as compared to dynamic testing.

Limitations of Static Testing

Below are some of the limitations of static testing:

1. **Detect Some Issues:** Static testing may not uncover all issues that could arise during runtime. Some defects may appear only during dynamic testing when the software runs.
2. **Depends on the Reviewer's Skills:** The effectiveness of static testing depends on the reviewer's skills, experience, and knowledge.
3. **Time-consuming:** Static testing can be time-consuming when working on large and complex projects.
4. **No Runtime Environment:** It is conducted without executing the code. This means it cannot detect runtime errors such as memory leaks, performance issues, etc.
5. **Prone to Human Error:** Static testing is prone to human error due to manual reviews and inspections techniques being used.