**UNIT IV – TESTING AND QUALITY IN SOFTWARE [9 hours]**

Types of Testing: Unit, Integration, System, Automated Testing using JUnit or PyTest, Introduction to Selenium (UI testing basics), Code Quality Tools: SonarLint/SonarQube basics , Importance of Testing in Real Projects

---

## TYPES OF TESTING

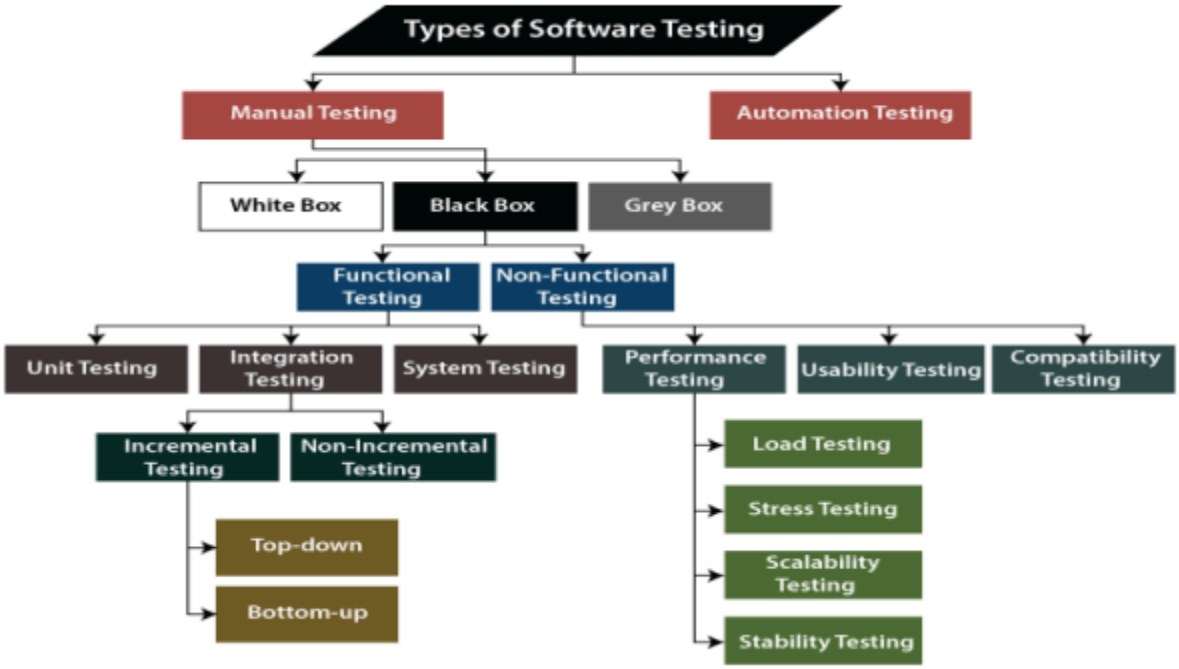
Software testing is a process of analyzing an application's functionality as per the customer prerequisite. If we want to ensure that our software is bug-free or stable, we must perform the various types of software testing because testing is the only method that makes our application bug-free.

**Various types of testing**

The categorization of software testing is a part of diverse testing activities, such as test strategy, test deliverables, a defined test objective, etc. And software testing is the execution of the software to find defects. The purpose of having a testing type is to confirm the AUT (Application Under Test). To start testing, we should have a requirement, application-ready, necessary resources available. To maintain accountability, we should assign a respective module to different test engineers.

The software testing mainly divided into two parts, which are as follows:

- Manual Testing
- Automation Testing



**What is Manual Testing?**

Testing any software or an application according to the client's needs without using any automation tool is known as manual testing. In other words, we can say that it is a procedure of verification and validation. Manual testing is used to verify the behavior of an application or software in contradiction of requirements specification. We do not require any precise knowledge of any testing tool to execute the manual test cases. We can easily prepare the test document while performing manual testing on any application.

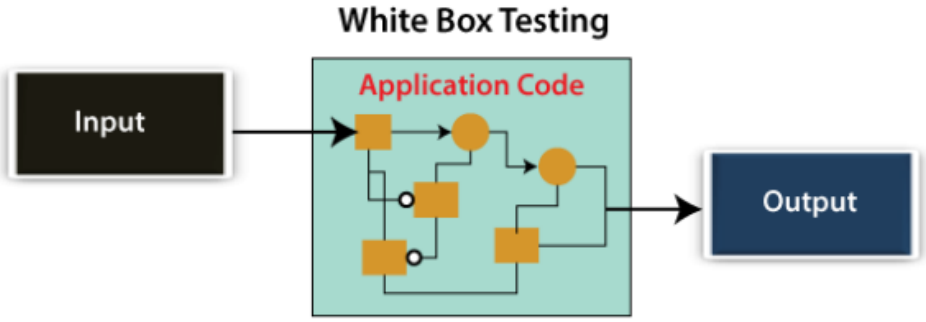
**Classification of Manual Testing**

In software testing, manual testing can be further classified into three different types of testing, which are as follows:

- White Box Testing
- Black Box Testing
- Grey Box Testing

**White Box Testing**

In white-box testing, the developer will inspect every line of code before handing it over to the testing team or the concerned test engineers. Subsequently, the code is noticeable for developers throughout testing; that's why this process is known as WBT (White Box Testing). In other words, we can say that the developer will execute the complete white-box testing for the particular software and send the specific application to the testing team. The purpose of implementing the white box testing is to emphasize the flow of inputs and outputs over the software and enhance the security of an application.




White box testing is also known as open box testing, glass box testing, structural testing, clear box testing, and transparent box testing.

**Black Box Testing**

Another type of manual testing is black-box testing. In this testing, the test engineer will analyze the software against requirements, identify the defects or bug, and sends it back to the development team. Then, the developers will fix those defects, do one round of White box testing, and send it to the testing team.

Here, fixing the bugs means the defect is resolved, and the particular feature is working according to the given requirement. The main objective of implementing the black box testing is to specify the business needs or the customer's requirements. In other words, we can say that black box testing is a process of checking the functionality of an application as per the customer requirement. The source code is not visible in this testing; that's why it is known as black-box testing.


**Types of Black Box Testing**

Black box testing further categorizes into two parts, which are as discussed below:

- Functional Testing
- Non-function Testing

**Functional Testing**

The test engineer will check all the components systematically against requirement specifications known as functional testing. Functional testing is also known as Component testing. In functional testing, all the components are tested by giving the value, defining the output, and validating the actual output with the expected value.

Functional testing is a part of black-box testing as its emphasis on application requirements rather than actual code. The test engineer has to test only the program instead of the system.

**Types of Functional Testing**

Just like another type of testing is divided into several parts, functional testing is also classified into various categories. The diverse types of Functional Testing contain the following:

- Unit Testing
- Integration Testing
- System Testing

**1. Unit Testing**

Unit testing is the first level of functional testing in order to test any software. Testing the module of an application independently or testing all the module functionality is called unit testing.

The primary objective of executing the unit testing is to confirm the unit components with their performance. Here, a unit is defined as a single testable function of

a software or an application. And it is verified throughout the specified application development phase.

Unit testing is the process of testing the smallest parts of the code, like a method in which we verify the code's correctness by running one by one. It's a key part of software development that improves code quality by testing each unit in isolation.

We can write unit tests for these code units and run them automatically every time when we make changes. If a test fails, it helps quickly to find and fix the issue. These tests check that the code works as expected based on the logic the developer intended. Unit testing promotes modular code, ensures better test coverage, and saves time by allowing developers to focus more on coding than manual testing.

**Unit testing strategies**

To create effective unit tests, follow these basic techniques to ensure all scenarios are covered:

- **Logic checks:** Verify if the system performs correct calculations and follows the expected path with valid inputs. Check all possible paths through the code are tested.
- **Boundary checks:** Test how the system handles typical, edge case, and invalid inputs. For example, if an integer between 3 and 7 is expected, check how the system reacts to a 5 (normal), a 3 (edge case), and a 9 (invalid input).
- **Error handling:** Check the system properly handles errors. Does it prompt for a new input, or does it crash when something goes wrong?
- **Object-oriented checks:** If the code modifies objects, confirm that the object's state is correctly updated after running the code.

**Example**

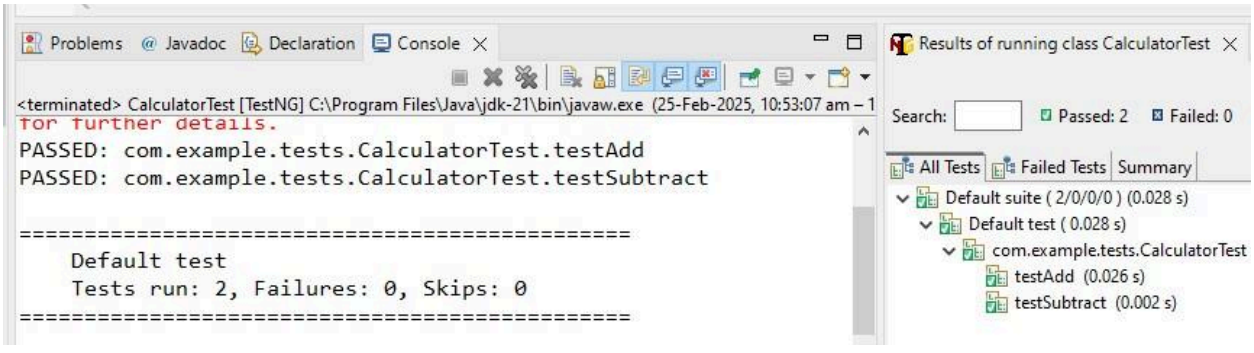
Here is the example of the java with unit test cases with the proper unit test code.

**Step 1. Create the Calculator Class.**

```
public class Calculator {
   // Method to add two numbers
   public int add(int a, int b) {
      return a + b;
   }
   // Method to subtract two numbers
   public int subtract(int a, int b) {
      return a - b;
   }
}
```

**Step 2. Create the TestNG Test Class.**

```
import org.testng.Assert;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
public class CalculatorTest {
    private Calculator calculator;
    // This method runs before each test method
    @BeforeMethod
    public void setUp() {
        calculator = new Calculator();
    }
    // Test for the 'add' method
    @Test
    public void testAdd() {
        int result = calculator.add(5, 3);
        // Assert that the result of 5 + 3 is 8
        Assert.assertEquals(result, 8, "Addition result is incorrect");
    }
    // Test for the 'subtract' method
    @Test
    public void testSubtract() {
        int result = calculator.subtract(5, 3);
        // Assert that the result of 5 - 3 is 2
        Assert.assertEquals(result, 2, "Subtraction result is incorrect");
    }
}
```




**Benefits of Unit Testing**

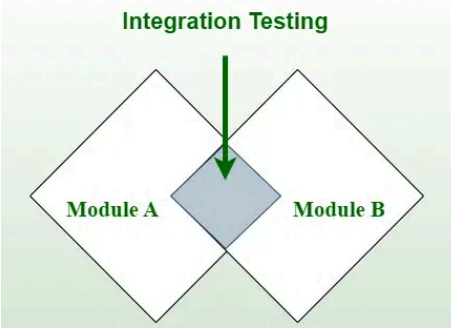
Here are the Unit testing benefits which used in the software development with many ways:

- **Early Detection of Issues:** Unit testing allows developers to detect and fix issues early in the development process before they become larger and more difficult to fix.
- **Improved Code Quality:** Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.
- **Increased Confidence:** Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.
- **Faster Development:** Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.
- **Better Documentation:** Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.
- **Facilitation of Refactoring:** Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.
- **Reduced Time and Cost:** Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.

**2. Integration Testing**

Integration Testing is a Software Testing Technique that focuses on verifying the interactions and data exchange between different components or modules of a Software Application. The goal of Integration Testing is to identify any problems or bugs that arise when different components are combined and interact with each other.

- It mainly tests the interface between two software units or modules. It focuses on determining the correctness of the interface. Once all the modules have been unit-tested, integration testing is performed.
- Integration testing can be done by picking module by module. This can be done so that there is a proper sequence to be followed.
- Exposing the defects is the major focus of the integration testing and the time of interaction between the integrated units.




**Types of Integration Testing**

Integration testing is also further divided into the following parts:

- Incremental Testing
- Non-Incremental Testing (Big Bang Integration Testing)

**Incremental Integration Testing**

Whenever there is a clear relationship between modules, we go for incremental integration testing. Suppose, we take two modules and analyze the data flow between them if they are working fine or not. If these modules are working fine, then we can add one more module and test again. And we can continue with the same process to get better results. In other words, we can say that incrementally adding up the modules and testing the data flow between the modules is known as Incremental integration testing.

**Types of Incremental Integration Testing**

Incremental integration testing can further classify as follows:

**1. Top-down Incremental Integration Testing**

**2. Bottom-up Incremental Integration Testing**

**3. Mixed Integration Testing**

Let's see a brief introduction of these types of integration testing:

**1. Top-down Incremental Integration Testing**

Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet integrated. In this integration testing, testing takes place from top to bottom. First, high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

**Advantages of Top-Down Integration Testing**

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

**Disadvantages of Top-Down Integration Testing**

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

**2. Bottom-up Incremental Integration Testing**

In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower-level modules.

**Advantages of Bottom-Up Integration Testing**

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

**Disadvantages of Bottom-Up Integration Testing**

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

**Mixed Integration Testing:**

A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In a top-down approach, testing can start only after the top-level module has been coded and unit tested. In the bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. It is also called the hybrid integration testing. Also, stubs and drivers are used in mixed integration testing.

**Advantages of Mixed Integration Testing**

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

**Disadvantages of Mixed Integration Testing**

- For mixed integration testing, it requires a very high cost because one part has a Top-down approach while another part has a bottom-up approach.
- This integration testing cannot be used for smaller systems with huge interdependence between different modules.

**Non-Incremental Integration Testing/ Big Bang Method**

It is the simplest integration testing approach, where all the modules are combined and the functionality is verified after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated.

**Advantages of Big-Bang Integration Testing**

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

**Disadvantages of Big-Bang Integration Testing**

- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
- High-risk critical modules are not isolated and tested on priority since all modules are tested at once.
- Not Good for long projects.
- High risk of integration problems that are difficult to identify and diagnose.

**3. System Testing**

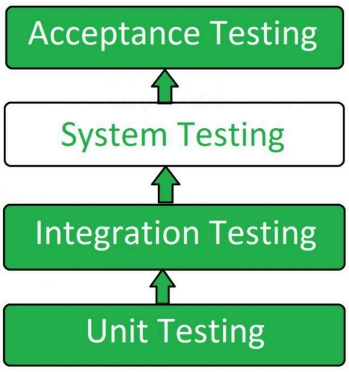
System Testing is a type of software testing that is performed on a completely integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input.

The goal of integration testing is to detect any irregularity between the units that are integrated. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested.

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or the context of both. System testing tests the design and behavior of the system and also the expectations of the customer.

It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS). System Testing is performed by a testing team that is independent of the development team and helps to test the quality of the system impartial.

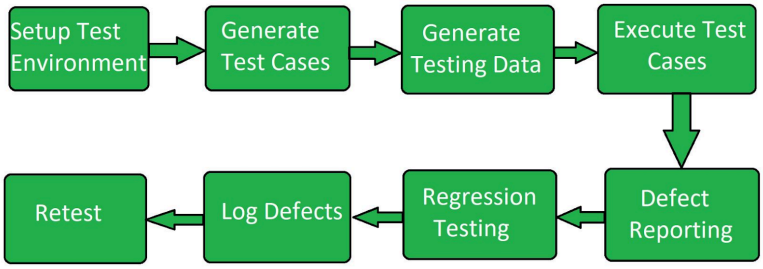
It has both functional and non-functional testing. System Testing is performed after the integration testing and before the acceptance testing.




**System Testing Process**

System Testing is performed in the following steps:

1. **Test Environment Setup:** Create testing environment for the better quality testing.
2. **Create Test Case:** Generate test case for the testing process.
3. **Create Test Data:** Generate the data that is to be tested.
4. **Execute Test Case:** After the generation of the test case and the test data, test cases are executed.
5. **Defect Reporting:** Defects in the system are detected.
6. **Regression Testing:** It is carried out to test the side effects of the testing process.
7. **Log Defects:** Defects are fixed in this step.
8. **Retest:** If the test is not successful then again the test is performed.




**Different Types of System Testing:**

- **Functional Testing:** This checks if the system's features work as expected and meet the defined requirements.
- **Performance Testing:** This tests how the system performs under different conditions, like high traffic or heavy use, to ensure it can handle the expected load.
- **Security Testing:** This ensures the system's security measures protect sensitive data from unauthorized access or attacks.

- **Compatibility Testing:** This makes sure the system works well across different hardware, software, and network environments.
- **Usability Testing:** This evaluates how easy and user-friendly the system is, making sure it provides a good experience for users.
- **Regression Testing:** This ensures that any new code or features don't break or negatively affect the system's existing functionality.
- **Acceptance Testing:** This tests the system at a high level to make sure it meets customer expectations and requirements before release.

**Advantages of System Testing**

Here are the Advantages of System Testing are follows:

- In System Testing The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing
- Verifies the overall functionality of the system.

**Disadvantages of System Testing**

Here are the Disadvantages of System Testing are follows:

- System Testing is a more time consuming process than other testing techniques since it checks the entire product or software.
- The cost for the testing will be high since it covers the testing of the entire software.
- It needs a good debugging tool otherwise the hidden errors will not be found.
- Can be time-consuming and expensive.
- Requires adequate resources and infrastructure.
- Can be complex and challenging, especially for large and complex systems.
- Dependent on the quality of requirements and design documents.

---