SOFTWARE ENGINEERING

UNIT- 4

Software Testing:

Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:

1. Verification: it refers to the set of tasks that ensure that the software correctly implements a specific function.

2. Validation: it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

Objectives:

- Finding defects which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product.

Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it's performing well and as per the specifications.

Software Testing Principles:

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. Here, in this section, we are going to learn about the seven essential principles of software testing.

Let us see the seven different testing principles, one by one:

1. Testing shows the presence of defects
2. Exhaustive Testing is not possible
3. Early Testing
4. Defect Clustering
5. Pesticide Paradox
6. Testing is context-dependent
7. Absence of errors fallacy

1. Testing shows the presence of defects

The test engineer will test the application to make sure that the application is bug or defects free. While doing testing, we can only identify that the application or software has any errors. The primary purpose of doing testing is to identify the numbers of unknown bugs with the help of various methods and testing techniques because the entire test should be traceable to the customer requirement, which means that to find any defects that might cause the product failure to meet the client's needs.

By doing testing on any application, we can decrease the number of bugs, which does not mean that the application is defect-free because sometimes the software seems to be bug-free while performing multiple types of testing on it. But at the time of deployment in the production server, if the end-user encounters those bugs which are not found in the testing process.

2. Exhaustive Testing is not possible

Sometimes it seems to be very hard to test all the modules and their features with effective and non- effective combinations of the inputs data throughout the actual testing process.

Hence, instead of performing the exhaustive testing as it takes boundless determinations and most of the hard work is unsuccessful. So, we can complete this type of variations according to the importance of the modules because the product timelines will not permit us to perform such type of testing scenarios.

3. Early Testing

Here early testing means that all the testing activities should start in the early stages of the software development life cycle's requirement analysis stage to identify the defects because if we find the bugs at an early stage, it will be fixed in the initial stage itself, which may cost us very less as compared to those which are identified in the future phase of the testing process.

4. Defect clustering

The defect clustering defined that throughout the testing process, we can detect the numbers of bugs which are correlated to a small number of modules. We have various reasons for this, such as the modules could be complicated; the coding part may be complex, and so on.

These types of software or the application will follow the Pareto Principle, which states that we can identify that approx. Eighty percent of the complication is present in 20 percent of the modules. With the help of this, we can find the uncertain modules, but this method has its difficulties if the same tests are performing regularly, hence the same test will not able to identify the new defects.

5. Pesticide paradox

This principle defined that if we are executing the same set of test cases again and again over a particular time, then these kinds of the test will not be able to find the new bugs in the software or the application. To get over these pesticide paradoxes, it is very significant to review all the test cases frequently. And the new and different tests are necessary to be written for the implementation of multiple parts of the application or the software, which helps us to find more bugs.

6. Testing is context-dependent

Testing is a context-dependent principle states that we have multiple fields such as e-commerce websites, commercial websites, and so on are available in the market. There is a definite way to test the commercial site as well as the e-commerce websites because every application has its own needs, features, and functionality. To check this type of application, we will take the help of various kinds of testing, different technique, approaches, and multiple methods. Therefore, the testing depends on the context of the application.

7. Absence of errors fallacy

Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free. But there is the chance when the application is tested beside the incorrect requirements, identified the flaws, and fixed them on a given period would not help as testing is done on the wrong specification, which does not apply to the client's requirements. The absence of error fallacy means identifying and fixing the bugs would not help if the application is impractical and not able to accomplish the client's requirements and needs.

Software Testability:

Software testability is the degree to which a software artifact (i.e. a software system, software module, requirements- or design document) supports testing in a given test context. If the testability of the software artifact is high, then finding faults in the system (if it has any) by means of testing is easier.

Testing is a critical stage of the software development lifecycle. The aim is to release bug-free, performant software that won't cost you a fortune in backend running costs. Clearly, making this process more efficient and effective will save you time and effort, and in the long run, will improve your profitability. This is one of the main drivers behind the switch to test automation. However, one important factor is often overlooked – software testability.

Testability, a property applying to empirical hypothesis, involves two components. The effort and effectiveness of software tests depends on numerous factors including:

- Properties of the software requirements
- Properties of the software itself (such as size, complexity and testability)
- Properties of the test methods used
- Properties of the development- and testing processes
- Qualification and motivation of the persons involved in the test process

Testability of software components:

The testability of software components (modules, classes) is determined by factors such as:

1. **Controllability:** The degree to which it is possible to control the state of the component under test (CUT) as required for testing.
2. **Observability:** The degree to which it is possible to observe (intermediate and final) test results.
3. **Isolate ability:** The degree to which the component under test (CUT) can be tested in isolation.
4. **Separation of concerns:** The degree to which the component under test has a single, well-defined responsibility.
5. **Understandability:** The degree to which the component under test is documented or self-explaining.
6. **Automatability:** The degree to which it is possible to automate testing of the component under test.
7. **Heterogeneity:** The degree to which the use of diverse technologies requires to use diverse test methods and tools in parallel.

Testability Hierarchy:

Based on the amount of test cases required to construct a complete test suite in each context (i.e., a test suite such that, if it is applied to the implementation under test, then we collect enough information to precisely determine whether the system is correct or incorrect according to some specification), a testability hierarchy with the following testability classes has been proposed:

- **Class I:** there exists a finite complete test suite.

- **Class II:** any partial distinguishing rate (i.e., any incomplete capability to distinguish correct systems from incorrect systems) can be reached with a finite test suite.
- **Class III:** there exists a countable complete test suite.
- **Class IV:** there exists a complete test suite.
- **Class V:** all cases.

Software Testability Measurement:

Software testability measurement refers to the activities and methods that study, analyze, and measure software testability during a software product life cycle. Once software is implemented, it is essential to make an assessment to finalize which software components are likely to be more difficult and time-consuming in testing due to their poor component testability.

There are numerous measures that can be taken to enhance testability as listed below:

- **Transparency:** Know in an out about the system, what it does and what it is supposed to do. Have a crystal-clear understanding. · Updated system documentation
- Comprehensive well-structured code
- Clarity in software requirements
- **Isolation:** Try techniques and mention codes by which parts of the system can be tested in isolation. Document each concern separately and make the results visible.
- Go for a test environment simulation: Strike a balance between representativeness and flexibility in Favour of the system and give in suggestions wherein you can opt for automated tests.

Verification & Validation:

Verification:

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing.

Validation:

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e., it checks what we are developing is the right product. it is validation of actual and expected product. Validation is the dynamic testing.

The difference between Verification and Validation is as follow:

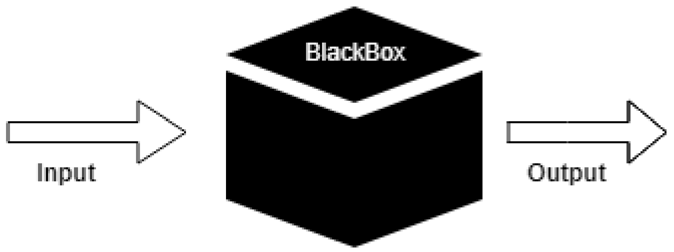
Verification	Validation

It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.
It does *not* include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

Black box testing:

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.



Generic steps of black box testing:

1. The black box test is based on the specification of requirements, so it is examined in the beginning.

2. In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
3. In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
4. The fourth phase includes the execution of all test cases.
5. In the fifth step, the tester compares the expected output against the actual output.
6. In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Advantages:

- Well suited and efficient for large code segments.
- Code access is not required.
- Clearly separates user's perspective from the developer's perspective through visibly defined roles.
- Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.

Disadvantages:

- Limited coverage, since only a selected number of test scenarios is actually performed.
- Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
- Blind coverage, since the tester cannot target specific code segments or error-prone areas.
- The test cases are difficult to design.

White Box Testing:

The box testing approach of software testing consists of black box testing and white box testing. We are discussing here white box testing which also known as **glass box testing, structural testing**, **clear box testing, open box testing** and **transparent box testing**.

It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term **'white box'** is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

Generic steps of white box testing

1. Design all test scenarios, test cases and prioritize them according to high priority number.
2. This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by various methods and operations and so on.
3. In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, interfaces are able to handle all types of data appropriately or not.
4. This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.
5. In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

Reasons for white box testing:

- It identifies internal security holes.
- To check the way of input inside the code.
- Check the functionality of conditional loops.
- To test function, object, and statement at an individual level.

Difference B/W Black box testing & White box testing:

S. No.	**Black Box Testing**	**White Box Testing**
1.	It 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.	It 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.
2.	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
3.	It is mostly done by software testers.	It is mostly done by software developers.
4.	No knowledge of implementation is needed.	Knowledge of implementation is required.
5.	It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
6.	It is a functional test of the software.	It is a structural test of the software.
7.	This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
8.	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
9.	It is the behavior testing of the software.	It is the logic testing of the software.
10.	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
11.	It is also called closed testing.	It is also called as clear box testing.

12.	It is least time consuming.	It is most time consuming.

Advantages of White box testing:

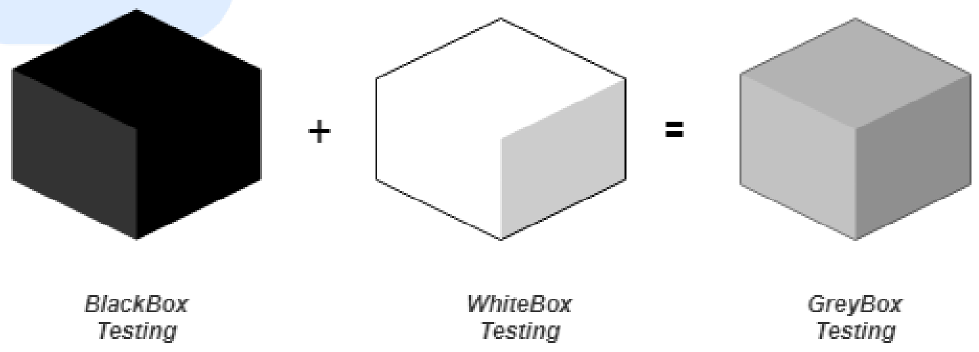
- White box testing optimizes code so hidden errors can be identified.
- Test cases of white box testing can be easily automated.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.

Disadvantages of White box testing:

- White box testing is too much time consuming when it comes to large-scale programming applications.
- White box testing is much expensive and complex.
- It can lead to production error because it is not detailed by the developers.
- White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

Grey Box Testing:

Grey box testing is a software testing method to test the software application with partial knowledge of the internal working structure. It is a combination of black box and white box testing because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.



Grey Box testing commonly identifies context-specific errors that belong to web systems. For example; while testing, if tester encounters any defect, then he makes changes in code to resolve the defect and then test it again in real time. It concentrates on all the layers of any complex software system to increase testing coverage. It gives the ability to test both presentation layer as well as internal coding structure. It is primarily used in integration testing and penetration testing.

Reasons to use Grey Box testing:

- It provides combined benefits of both Blackbox testing and Whitebox testing.
- It includes the input values of both developers and testers at the same time to improve the overall quality of the product.
- It reduces time consumption of long process of functional and non-functional testing.
- It gives sufficient time to the developer to fix the product defects.
- It includes user point of view rather than designer or tester point of view.
- It involves examination of requirements and determination of specifications by user point of view deeply.

Levels of Software Testing:

Software level testing can be majorly classified into 4 levels:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

Unit Testing:

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with its performance.

A unit is a single testable part of a software system and tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. White box testing approach used for unit testing and usually done by the developers.

Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or one by one, and this process is known as **Unit testing** or **components testing**.

Unit Testing Tools:

We have various types of unit testing tools available in the market, which are as follows:

- NUnit
- JUnit
- PHPunit
- Parasoft Jtest
- EMMA

Unit Testing Techniques:

Unit testing uses all white box testing techniques as it uses the code of software application:

- Data flow Testing
- Control Flow Testing
- Branch Coverage Testing
- Statement Coverage Testing
- Decision Coverage Testing

Advantages:

- Unit testing uses module approach due to that any part can be tested without waiting for completion of another parts testing.
- The developing team focuses on the provided functionality of the unit and how functionality should look in unit test suits to understand the unit API.
- Unit testing allows the developer to refactor code after a number of days and ensure the module still working without any defect.

Disadvantages:

- It cannot identify integration or broad level error as it works on units of the code.
- In the unit testing, evaluation of all execution paths is not possible, so unit testing is not able to catch each and every error in a program.
- It is best suitable for conjunction with other testing activities

Integration testing:

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

Integration test approaches:

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

1. Big-Bang Integration Testing:

It is the simplest integration testing approach, where all the modules are combined and the functionality is verified after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. 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. So, debugging errors reported during big bang integration testing is very expensive to fix.

Advantages:

- It is convenient for small systems.

Disadvantages:

- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
- High risk critical modules are not isolated and tested on priority since all modules are tested at once.

2. Bottom-Up 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:

- In bottom-up testing, no stubs are required.
- A principal advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:

- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystems.

3. Top-Down Integration Testing:

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

Advantages:

- Separately debugged module.
- Few or no drivers needed.
- It is more stable and accurate at the aggregate level.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

4. Mixed Integration Testing:

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

Advantages:

- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.

Disadvantages:

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

System Testing:

System Testing includes testing of a fully integrated software system. Generally, a computer system is made with the integration of software. The software is developed in units and then interfaced with other software and hardware to create a complete computer system.

In other words, a computer system consists of a group of software to perform the various tasks, but only software cannot perform the task; for that software must be interfaced with compatible hardware. System testing is a series of different type of tests with the purpose to exercise and examine the full working of an integrated software computer system against requirements.

To check the end-to-end flow of an application or the software as a user is known as System testing. In this, we navigate (go through) all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.

It is **end-to-end testing** where the testing environment is similar to the production environment.

Types of System Testing:

System testing is divided into more than 50 types, but software testing companies typically uses some of them. These are listed below:

Regression Testing:

Regression testing is performed under system testing to confirm and identify that if there's any defect in the system due to modification in any other part of the system. It makes sure, any changes done during the development process have not introduced a new defect and also gives assurance; old defects will not exist on the addition of new software over the time.

Load Testing:

Load testing is performed under system testing to clarify whether the system can work under real-time loads or not.

Functional Testing:

Functional testing of a system is performed to find if there's any missing function in the system. Tester makes a list of vital functions that should be in the system and can be added during functional testing and should improve quality of the system.

Recovery Testing:

Recovery testing of a system is performed under system testing to confirm reliability, trustworthiness, accountability of the system and all are lying on recouping skills of the system. It should be able to recover from all the possible system crashes successfully.

In this testing, we will test the application to check how well it recovers from the crashes or disasters.

Migration Testing:

Migration testing is performed to ensure that if the system needs to be modified in new infrastructure so it should be modified without any issue.

Usability Testing:

The purpose of this testing to make sure that the system is well familiar with the user and it meets its objective for what it supposed to do.

Software and Hardware Testing:

This testing of the system intends to check hardware and software compatibility. The hardware configuration must be compatible with the software to run it without any issue. Compatibility provides flexibility by providing interactions between hardware and software.

Advantages of System Testing:

- Verifies the system against the business, functional and technical requirements of the end users.
- It helps in getting maximum bugs before acceptance testing.
- System testing increases the confidence level of the team in the product before the product goes for acceptance testing.
- It is the first testing level in which the whole system is under test from end to end. So, it helps in finding important defects which, unit and integration testing could not detect.
- This testing phase uses the test environment which is similar to the real business environment or production environment. Consequently, it helps in boosting the confidence of users into the product.
- It is a black box testing hence testers do not need programming knowledge to perform it.

Disadvantages of System Testing:

- This testing is time consuming process than another testing techniques since it checks the entire product or software.
- The cost for the testing will be high since it covers the testing of entire software.
- It needs good debugging tool otherwise the hidden errors will not be found.

Acceptance Testing:

Acceptance Testing is a method of software testing where a system is tested for acceptability. The major aim of this test is to evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not.

It is a formal testing according to user needs, requirements and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers or other authorized entities to determine whether to accept the system or not.

Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.

Types of Acceptance Testing:

1. User Acceptance Testing (UAT)
2. Business Acceptance Testing (BAT)
3. Contract Acceptance Testing (CAT)
4. Regulations Acceptance Testing (RAT)
5. Operational Acceptance Testing (OAT)
6. Alpha Testing
7. Beta Testing

Use of Acceptance Testing:

- To find the defects missed during the functional testing phase.
- How well the product is developed.
- A product is what actually the customers need.
- Feedback helps in improving the product performance and user experience.
- Minimize or eliminate the issues arising from the production.

Advantages of Acceptance Testing:

- This testing helps the project team to know the further requirements from the users directly as it involves the users for testing.
- Automated test execution.
- It brings confidence and satisfaction to the clients as they are directly involved in the testing process.
- It is easier for the user to describe their requirement.
- It covers only the Black-Box testing process and hence the entire functionality of the product will be tested.

Disadvantages of Acceptance Testing:

- Users should have basic knowledge about the product or application.
- Sometimes, users don’t want to participate in the testing process.
- The feedback for the testing takes long time as it involves many users and the opinions may differ from one user to another user.
- Development team is not participated in this testing process.