SE

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Testing Objectives

Software Testing has different goals and objectives.The major objectives of Software testing are as follows:

- 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.

While determining the test coverage the test cases should be designed well with maximum possibilities of finding the errors or bugs. The test cases should be very effective. This objective can be measured by the number of defects reported per test cases. Higher the number of the defects reported the more effective are the test cases.

Once the delivery is made to the end users or the customers they should be able to operate it without any complaints. In order to make this happen the tester should know as how the customers are going to use this product and accordingly they should write down the test scenarios and design the test cases. This will help a lot in fulfilling all the customer's requirements.

Software testing makes sure that the testing is being done properly and hence the system is ready for use. Good coverage means that the testing has been done to cover the various areas like functionality of the application, compatibility of the application with the OS, hardware and different types of browsers, performance testing to test the performance of the application and load testing to make sure that the system is reliable and should not crash or there should not be any blocking issues. It also determines that the application can be deployed easily to the machine and without any resistance. Hence the application is easy to install, learn and use.

Principles of Testing

There are seven principles of testing. They are as follows:

1. **Testing shows presence of defects:** Testing can show the defects are present, but cannot prove that there are no defects. Even after testing the application or product thoroughly we cannot say that the product is 100% defect free. Testing always reduces the number of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.
2. **Exhaustive testing is impossible:** Testing everything including all combinations of inputs and preconditions is not possible. So, instead of doing the exhaustive testing we can use risks and priorities to focus

testing efforts. For example: In an application in one screen there are 15 input fields, each having 5 possible values, then to test all the valid combinations you would need 30 517 578 125 (515) tests. This is very unlikely that the project timescales would allow for this number of tests. So, accessing and managing risk is one of the most important activities and reason for testing in any project.

3. **Early testing:** In the software development life cycle testing activities should start as early as possible and should be focused on defined objectives.
4. **Defect clustering:** A small number of modules contains most of the defects discovered during pre-release testing or shows the most operational failures.
5. **Pesticide paradox:** If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new bugs. To overcome this "Pesticide Paradox", it is really very important to review the test cases regularly and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.
6. **Testing is context dependent:** Testing is basically context dependent. Different kinds of sites are tested differently. For example, safety – critical software is tested differently from an e-commerce site.
7. **Absence – of – errors fallacy:** If the system built is unusable and does not fulfill the user's needs and expectations then finding and fixing defects does not help.

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.

Testability os often thought of as an extrinsic property which results from interdependency of the software to be tested and the test goals, test methods used, and test resources (i.e., the test context). Even though testability cannot be measured directly (such as software size) it should be considered an intrinsic property of a software artifact because it is highly correlated with other key software qualities such as encapsulation, coupling, cohesion, and redundancy.

The correlation of 'testability' to good design can be observed by seeing that code that has weak cohesion, tight coupling, redundancy and lack of encapsulation is difficult to test.

A lower degree of testability results in increased test effort. In extreme cases a lack of testability may hinder testing parts of the software or software requirements at all.

In order to link the testability with the difficulty to find potential faults in a system (if they exist) by testing it, a relevant measure to assess the testability is how many test cases are needed in each case to form a complete test suite (i.e. a test suite such that, after applying all test cases to the system, collected outputs will let us unambiguously determine whether the system is correct or not according to some specification). If this size is small, then the testability is high. Based on this measure, a testability hierarchy has been proposed.

(1)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.

It has been proved that each class is strictly included into the next. For instance, testing when we assume that the behavior of the implementation under test can be denoted by a deterministic finite-state machine for some known finite sets of inputs and outputs and with some known number of states belongs to Class I (and all subsequent classes). However, if the number of states is not known, then it only belongs to all classes from Class II on. If the implementation under test must be a deterministic finite-state machine failing the specification for a single trace (and its continuations), and its number of states is unknown, then it only belongs to classes from Class III on. Testing temporal machines where transitions are triggered if inputs are produced within some real-bounded interval only belongs to classes from Class IV on, whereas testing many non-deterministic systems only belongs to Class V (but not all, and some even belong to Class I). The inclusion into Class I does not require the simplicity of the assumed computation model, as some testing cases involving implementations written in any programming language, and testing implementations defined as machines depending on continuous magnitudes, have been proved to be in Class I. Other elaborated cases, such as the testing framework by Matthew Hennessyunder must semantics, and temporal machines with rational timeouts, belong to Class II.

(2)Testability of requirements

Requirements need to fulfill the following criteria in order to be testable:

- consistent
- complete
- unambiguous
- quantitative (a requirement like "fast response time" can not be verification/verified)
- verification/verifiable in practice (a test is feasible not only in theory but also in practice with limited resources)

What is Test case?

A test case is a document, which has a set of test data, preconditions, expected results and postconditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test Case acts as the starting point for the test execution, and after applying a set of input values; the application has a definitive outcome and leaves the system at some end point or also known as execution postcondition.

Typical Test Case Parameters:

- Test Case ID
- Test Scenario
- Test Case Description
- Test Steps
- Prerequisite
- Test Data
- Expected Result
- Test Parameters
- Actual Result

- Environment Information
- Comments

Example:

Let us say that we need to check an input field that can accept maximum of 10 characters.

While developing the test cases for the above scenario, the test cases are documented the following way. In the below example, the first case is a pass scenario while the second case is a FAIL.

Scenario	Test Step	Expected Result	Actual Outcome
Verify that the input field that can accept maximum of 10 characters	Login to application and key in 10 characters	Application should be able to accept all 10 characters.	Application accepts all 10 characters.
Verify that the input field that can accept maximum of 11 characters	Login to application and key in 11 characters	Application should NOT accept all 11 characters.	Application accepts all 10 characters.

If the expected result doesn't match with the actual result, then we log a defect. The defect goes through the defect life cycle and the testers address the same after fix.

Test case Design Technique

Following are the typical design techniques in software engineering:

1. Deriving test cases directly from a requirement specification or black box test design technique. The Techniques include:

- **Boundary Value Analysis (BVA)**
- **Equivalence Partitioning (EP)**

- **Decision Table Testing**
- **State Transition Diagrams**
- **Use Case Testing**

2. Deriving test cases directly from the structure of a component or system:

- **Statement Coverage**
- **Branch Coverage**
- **Path Coverage**
- **LCSAJ Testing**

3. Deriving test cases based on tester's experience on similar systems or testers intuition:

- **Error Guessing**
- **Exploratory Testing**

Types of Testing

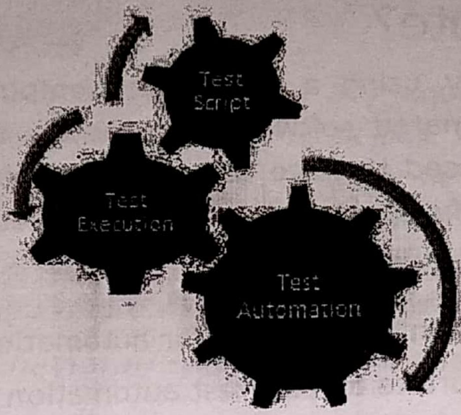
Manual Testing

Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

Automation Testing

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.



Apart from regression testing, automation testing is also used to test the application from load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money in comparison to manual testing.

What is Automate?

It is not possible to automate everything in a software. The areas at which a user can make transactions such as the login form or registration forms, any area where large number of users can access the software simultaneously should be automated.

Furthermore, all GUI items, connections with databases, field validations, etc. can be efficiently tested by automating the manual process.

When to Automate?

Test Automation should be used by considering the following aspects of a software:

- Large and critical projects
- Projects that require testing the same areas frequently
- Requirements not changing frequently
- Accessing the application for load and performance with many virtual users
- Stable software with respect to manual testing
- Availability of time

How to Automate?

Automation is done by using a supportive computer language like VB scripting and an automated software application. There are many tools available that can be used to write automation scripts. Before mentioning the tools, let us identify the process that can be used to automate the testing process:

- Identifying areas within a software for automation
- Selection of appropriate tool for test automation
- Writing test scripts
- Development of test suits
- Execution of scripts
- Create result reports
- Identify any potential bug or performance issues

Software Testing Tools

The following tools can be used for automation testing:

- HP Quick Test Professional
- Selenium
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LoadRunner
- Visual Studio Test Professional
- WATIR

Testing Method

There are different methods that can be used for software testing. This chapter briefly describes the methods available.

Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

It is also called as Behavioral/Specification-Based/Input-Output Testing

Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure. This can be applied to every level of software testing such as Unit, Integration, System and Acceptance Testing.

Testers create test scenarios/cases based on software requirements and specifications. Tester performs testing only on the functional part of an application to make sure the behavior of the software is as expected.

The tester passes input data to make sure whether the actual output matches the expected output.

Black Box Testing Techniques:

It is used to minimize the number of possible test cases to an optimum level while maintaining reasonable test coverage

Equivalence Partitioning: Equivalence Partitioning is also known as Equivalence Class Partitioning. In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be proposed in the same way. Hence selecting one input from each group to design the test cases. [illegible]
[illegible].

Boundary Value Analysis: Boundary value analysis (BVA) is based on testing the boundary values of valid and invalid partitions. The Behavior at

Boundary value testing is focused on the values at boundaries. This techiniaue determines whether a certain range of values are acceptable by the system or not. It reduce test case. It is suitable for the systems where input is within certain ranges the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects. ~~[illegible] detailed post on boundary value analysis.~~

Decision Table: Decision Table is aka Cause-Effect Table. This test technique is appropriate for functionalities which has logical relationships between inputs (if-else logic). In Decision table technique, we deal with combinations of inputs. To identify the test cases with decision table, we consider conditions and actions. We take conditions as inputs and actions as outputs. Click here to see detailed post on decision table.

State Transition: Using state transition testing, we pick test cases from an application where we need to test different system transitions. We can apply this when an application gives a different output for the same input, depending on what has happened in the earlier state. ~~[illegible]~~ ~~detailed post on state transition technique~~

A decision table puts Causes and their effects in a matrix. There is Unique Combination in each column.

Types of Black Box Testing:

Functionality Testing: In simple words, what the system actually does is functional testing. It is related to functional requirment of a System, is done by S/w Testers.

Non-functionality Testing: In simple words, how well the system performs is non-functionality testing. It is not related to testing of a specific functionality but not functional requirment

The following table lists the advantages and disadvantages of black-box testing.

Advantages	Disadvantages
• Well suited and efficient for large code segments.	• Limited coverage, since only a selected number

Such as performance, Scability, Usability

• 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.	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

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

It is also called as Glass Box, Clear Box, Structural Testing.

White Box Testing is based on applications internal code structure. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. This testing usually done at the unit level.

White Box Testing Techniques:

1. **Statement Test**
   All the statements with in the code must have a test case associated with it such that each statement must be executed at least once during the testing cycle.
2. **Branch Condition Test**
   All the conditions in a specific decision must be tested for proper working at least once.
3. **Decision Test**
   All the decision directions must be executed at least once during the testing life cycle.
4. **Data Flow Test**
   This will ensure that all the variables and data that are used with in the the system are tested by passing the specific variables through each possible calculation.

The following table lists the advantages and disadvantages of white-box testing.

Advantages	Disadvantages
• As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. • It helps in optimizing the code. • Extra lines of code can be removed which can bring	• Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased. • Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested. • It is difficult to maintain white-box testing, as it requires

in hidden defects. • Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	specialized tools like code analyzers and debugging tools.

Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages	**Disadvantages**
• Offers combined benefits of black-box and white-box testing wherever possible. • Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications. • Based on the limited information available, a grey-box tester can design excellent test scenarios especially around	• Since the access to source code is not available, the ability to go over the code and test coverage is limited. • The tests can be redundant if the software designer has already run a test case. • Testing every possible input stream is unrealistic because it would take an unreasonable amount of

communication protocols and data type handling. • The test is done from the point of view of the user and not the designer.	time; therefore, many program paths will go untested.

A Comparison of Testing Methods

The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations - Internal behavior of the application is	Testing is done on the basis of high-level database diagrams and	Internal workings are fully known and the tester can design test data

unknown.	data flow diagrams.	accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

Verification and validation

Verification

The process of evaluating software to determine whether the products of agiven development phase satisfy the conditions imposed at the start of that phase.

Verification is a static practice of verifying documents, design, code and program. It includes all the activities associated with producing high quality software: inspection, design analysis and specification analysis. It is a relatively objective process.

Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful. Verification is concerned with whether the system is well-engineered and error-free.

Validation

The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

Validation is the process of evaluating the final product to check whether the software meets the customer expectations and requirements. It is a dynamic mechanism of validating and testing the actual product.

Verification	Validation
1. **Verification** is a static practice of verifying documents, design, code and program.	1. **Validation** is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.
3. It is human based checking of documents and files.	3. It is computer based execution of program.
4. **Verification** uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. **Validation** uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is High Level Exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. **Verification** is done by QA team to ensure that the software is as per the specifications in the SRS document.	8. **Validation** is carried out with the involvement of testing team.
9. It generally comes first-done before validation.	9. It generally follows after **verification**.

Unit testing
Integration testing
Validation testing
System testing