SOFTWARE TESTING

Software testing is widely used technology because it is compulsory to test each and every software before deployment.

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects. Here are the benefits of using software testing:

- **Cost-Effective**: It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.
- **Security**: It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.
- **Product quality**: It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.
- **Customer Satisfaction**: The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

Types of Testing

There are many different types of testing. Here is a quick breakdown of the most common testing types:

- Accessibility testing
- Acceptance testing
- Black box testing
- End to end testing
- Functional testing
- Interactive testing
- Integration testing
- Load testing
- Non functional testing
- Performance testing
- Regression testing
- Sanity testing
- Security testing
- Single user performance testing
- Smoke testing
- Stress testing
- Unit testing
- White-box testing and many more...

Many of these types of testing can be done manually — or they can be automated.

Accessibility Testing

Accessibility testing is the practice of ensuring your mobile and web apps are working and usable for users without and with disabilities such as vision impairment, hearing disabilities, and other physical or cognitive conditions.

Acceptance Testing

Acceptance testing ensures that the end-user (customers) can achieve the goals set in the business requirements, which determines whether the software is acceptable for delivery or not. It is also known as user acceptance testing (UAT).

Black Box Testing

Black box testing involves testing against a system where the code and paths are invisible.

End to End Testing

End to end testing is a technique that tests the application's workflow from beginning to end to make sure everything functions as expected.

Functional Testing

Functional testing checks an application, website, or system to ensure it's doing exactly what it's supposed to be doing.

Interactive Testing

Also known as manual testing, interactive testing enables testers to create and facilitate manual tests for those who do not use automation and collect results from external tests.

Integration Testing

Integration testing ensures that an entire, integrated system meets a set of requirements. It is performed in an integrated hardware and software environment to ensure that the entire system functions properly.

Load Testing

This type of non-functional software testing process determines how the software application behaves while being accessed by multiple users simultaneously.

Non Functional Testing

Non functional testing verifies the readiness of a system according to nonfunctional parameters (performance, accessibility, UX, etc.) which are never addressed by functional testing.

Performance Testing

Performance testing examines the speed, stability, reliability, scalability, and resource usage of a software application under a specified workload.

Regression Testing

Regression testing is performed to determine if code modifications break an application or consume resources.

Sanity Testing

Performed after bug fixes, sanity testing determines that the bugs are fixed and that no further issues are introduced to these changes.

Security Testing

Security testing unveils the vulnerabilities of the system to ensure that the software system and application are free from any threats or risks. These tests aim to find any potential flaws and weaknesses in the software system that could lead to a loss of data, revenue, or reputation per employees or outsides of a company.

Single User Performance Testing

Single user performance testing checks that the application under test performs fine according to specified threshold without any system load. This benchmark can be then used to define a realistic threshold when the system is under load.

Smoke Testing

This type of software testing validates the stability of a software application, it is performed on the initial software build to ensure that the critical functions of the program are working.

Stress Testing

Stress testing is a software testing activity that tests beyond normal operational capacity to test the results.

Unit Testing

Unit testing is the process of checking small pieces of code to ensure that the individual parts of a program work properly on their own, speeding up testing strategies and reducing wasted tests.

White Box Testing

White box testing involves testing the product's underlying structure, architecture, and code to validate input-output flow and enhance design, usability, and security.

Ways to Test

There are different types of testing that may be used to test a software during SDLC.

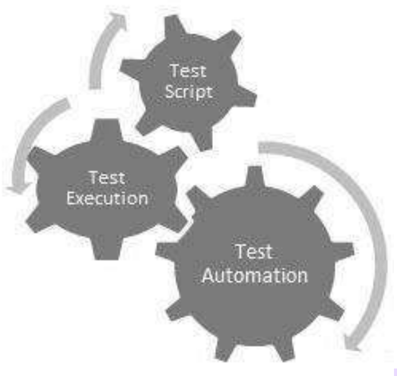
Manual Testing

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 behaviour 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.

Continuous testing

Continuous testing goes even further, applying the principles of automated testing in a scaled, continuous manner to achieve the most reliable test coverage for an enterprise. Keep reading to learn more about the differences between automated testing vs. manual testing and how continuous testing fits in.

Manual Vs Automation Testing

Aspects	Automation testing	Manual testing
Definition	When an application or software is tested with the help of some tools is known as automation testing. Whenever multiple releases or multiple regression cycle is going on the application or software, we will go for automation testing.	It is a type of software testing, which is done by the test Engineer to check the functionality of an application based on the customer requirement.
Reliability	It is reliable because it tests the application with the help of tools and test scripts.	It is not reliable because there is a possibility of human error, which may not be delivered the bug-free application.
Reused	The script can be reused across multiple releases.	It could be possible when the test case only needs to run once or twice.

Batch Execution	Batch execution is possible using automation testing because all the written scripts can be executed parallelly or simultaneously.	Batch execution is not possible in manual testing.
Time-saving	The execution is always faster than the manual; that's why the automation testing process is time-saving.	It is time consuming due to the usage of the human resources.
Investment	While using the Automation tool, investment is required.	Human resources needed investment.
Performance testing	To test the performance of the application with the help of load and stress testing, automation test engineer needs to perform Performance Testing.	In manual testing, performance testing is not possible.
Programming knowledge	Without having an understanding of programming language, we cannot write the test script.	There is no need-to-know programming language but should have the product knowledge to write the test case.
Framework	The automation test engineer can use the different types of frameworks like **Data driven, Hybrid, modular driven, and keyword-driven** to faster the automation process.	There is no need for a framework while using manual testing.
Operating system compatibility	Automation testing can also be performed on different systems with different operating system platforms and various programming languages.	Operating system compatibility is not possible in manual testing because the different tester is required to perform such tasks.
Regression testing	Whenever the code changes happen due to the enhancement of the release, then automation test engineer performs the regression testing.	When the test engineer executes the test case for the first time, it may be useful, but there is a possibility that it will not catch the regression bugs because of changing requirements frequently.

Test Plan, Test Coverage, Test Cases

1. Test Plan

A Test Plan refers to a detailed document that catalogs the test strategy, objectives, schedule, estimations, deadlines, and the resources required for completing that particular project. Think of it as a blueprint for running the tests needed to ensure the software is working properly – controlled by test managers.

A well-crafted test plan is a dynamic document that changes according to progressions in the project and stays current at all times. It is the point of reference, based on which testing activities are executed and coordinated among a QA team.

The test plan is also shared with Business Analysts, Project Managers, Dev teams, and anyone else associated with the project, This mainly offers transparency into QA activities so that all stakeholders know how the software will be tested.

The plan is built by QA managers or leads based on input from QA (and sometimes, non-QA) team members. Creating it should not take more than 1/3rd of the time allocated for the entire project.

Why are Test Plans important?

- They help individuals outside the QA teams (developers, business managers, customer-facing teams) understand exactly how the website or app will be tested.
- They offer a clear guide for QA engineers to conduct their testing activities.
- They detail aspects such as test scope, test estimation, strategy, and so on. Collating all this information into a single document makes it easier to review by management personnel or to re-use for other projects.

Components of a Test Plan

- **Scope**: Details the objectives of the particular project. Also, it details user scenarios to be used in tests. If necessary, the scope can specify what scenarios or issues the project will not cover.
- **Schedule:** Details start dates and deadlines for testers to deliver results.
- **Resource Allocation**: Details which tester will work on which test.
- **Environment**: Details the nature, configuration, and availability of the test environment.
- **Tools:** Details what tools are to be used for testing, bug reporting, and other relevant activities.
- **Defect Management**: Details how bugs will be reported, to whom and what each bug report needs to be accompanied by. For example, should bugs be reported with screenshots, text logs, or videos of their occurrence in the code?
- **Risk Management**: Details what risks may occur during software testing, and what risks the software itself may suffer if released without sufficient testing.

- **Exit Parameters**: Details when testing activities must stop. This part describes the results that are expected from the QA operations, giving testers a benchmark to compare actual results to.

Test Planning Activities:

- To determine the scope and the risks that need to be tested and that are NOT to be tested.
- Documenting Test Strategy.
- Making sure that the testing activities have been included.
- Deciding Entry and Exit criteria.
- Evaluating the test estimate.
- Planning when and how to test and deciding how the test results will be evaluated and defining test exit criterion.
- The Test artefacts delivered as part of test execution.
- Defining the management information, including the metrics required and defect resolution and risk issues.
- Ensuring that the test documentation generates repeatable test assets.

2. Test Coverage

Test coverage is defined as a metric in Software Testing that measures the amount of testing performed by a set of tests. It will include gathering information about which parts of a program are executed when running the test suite to determine which branches of conditional statements have been taken.

In simple terms, it is a technique to ensure that your tests are testing your code or how much of your code you exercised by running the test.

What Test Coverage does?

- Finding the area of a requirement not implemented by a set of test cases
- Helps to create additional test cases to increase coverage
- Identifying a quantitative measure of test coverage, which is an indirect method for quality check
- Identifying meaningless test cases that do not increase coverage

How Test Coverage can be accomplished?

- Test coverage can be done by exercising the static review techniques like peer reviews, inspections, and walkthrough
- By transforming the ad-hoc defects into executable test cases
- At code level or unit test level, test coverage can be achieved by availing the automated code coverage or unit test coverage tools
- Functional test coverage can be done with the help of proper test management tools

Benefits of Test Coverage

- It can assure the quality of the test
- It can help identify what portions of the code were actually touched for the release or fix

- It can help to determine the paths in your application that were not tested
- Prevent Defect leakage
- Time, scope and cost can be kept under control
- Defect prevention at an early stage of the project lifecycle
- It can determine all the decision points and paths used in the application, which allows you to increase test coverage
- Gaps in requirements, test cases and defects at the unit level and code level can be found in an easy way

Differences Between Code Coverage and Test Coverage

Code coverage and test coverage are measurement techniques which allow you to assess the quality of your application code.

Here, are some critical differences between booths of these coverage methods:

Parameters	Code Coverage	Test Coverage
Definition	Code coverage term used when application code is exercised when an application is running.	Test coverage means overall test-plan.
Goal	Code coverage metrics can help the team monitor their automated tests.	Test coverage is given details about the level to which the written coding of an application has been tested.
Subtypes	Code coverage divided with subtypes like statement coverage, condition coverage, Branch coverage, Toogle coverage, FSM coverage.	No subtype of Test coverage method.

3. Test Cases

In the simplest form, a test case is a set of conditions or variables under which a tester determines whether the software satisfies requirements and functions properly.

A test case is a single executable test which a tester carries out. It guides them through the steps of the test. You can think of a test case as a set of step-by-step instructions to verify something behaves as it is required to behave.

Why test cases are important

Test cases define what must be done to test a system, including the steps executed in the system, the input data values that are entered into the system and the results that are expected throughout test case execution. Using test cases allows developers and testers to discover errors that may have occurred during development or defects that were missed during ad hoc tests.

The benefits of an effective test case include:

- Guaranteed good test coverage.
- Reduced maintenance and software support costs.
- Reusable test cases.
- Confirmation that the software satisfies end-user requirements.
- Improved quality of software and user experience.
- Higher quality products lead to more satisfied customers.
- More satisfied customers will increase company profits.

Overall, writing and using test cases will lead to business optimization. Clients are more satisfied, customer retention increases, the costs of customer service and fixing products decreases, and more reliable products are produced, which improves the company's reputation and brand image.

Test cases have a few integral parts that should always be present in fields. However, every test case can be broken down into 8 basic steps.

Step 1: Test Case ID

Test cases should all bear unique IDs to represent them. In most cases, following a convention for this naming ID helps with organization, clarity, and understanding.

Step 2: Test Description

This description should detail what unit, feature, or function is being tested or what is being verified.

Step 3: Assumptions and Pre-Conditions

This entails any conditions to be met before test case execution. One example would be requiring a valid Outlook account for a login.

Step 4: Test Data

This relates to the variables and their values in the test case. In the example of an email login, it would be the username and password for the account.

Step 5: Steps to be Executed

These should be easily repeatable steps as executed from the end user's perspective. For instance, a test case for logging into an email server might include these steps:

1. Open email server web page.
2. Enter username.
3. Enter password.
4. Click "Enter" or "Login" button.

Step 6: Expected Result

This indicates the result expected after the test case step execution. Upon entering the right login information, the expected result would be a successful login.

Step 7: Actual Result and Post-Conditions

As compared to the expected result, we can determine the status of the test case. In the case of the email login, the user would either be successfully logged in or not. The post-condition is what happens as a result of the step execution such as being redirected to the email inbox.

Step 8: Pass/Fail

Determining the pass/fail status depends on how the expected result and the actual result compare to each other.

Same result = Pass
Different results = Fail

Before writing a test case, QA engineers and testing team members should first determine the scope and purpose of the test. This includes understanding the system features and user requirements as well as identifying the testable requirements.

Next, testers should define how the software testing activities are performed. This process starts by identifying effective test case scenarios -- or functionality that can be tested. In order to identify test case scenarios, testers must understand the functional requirements of the system.

Once the test case scenarios have been identified, the non-functional requirements must be defined. Non-function requirements include operating systems, security features and hardware requirements. Prerequisites for the test should also be pointed out.

The next step is to define the test case framework. Test cases typically analyse compatibility, functionality, fault tolerance, user interface (UI) and the performance of different elements.

Once these steps have been completed, the tester can begin writing the test case.