

QA Manual



Prepared By: Milan Karki

Introduction To Software Testing	4
Origin of Software Testing	4
Objectives of Testing	4
Requirements for testing	5
Functional and Non-Functional requirements	5
Advantages and Disadvantages of testing	6
Characteristics of Testing	6
Difference between Testing and Debugging	7
Quality Assurance, Quality Control, and Testing	8
Verification and Validation	8
Errors, Defects, and Failures	8
STLC VS SDLC	9
SDLC	9
Phases of SDLC	9
STLC	11
Difference between SDLC and STLC	13
Types of Testing	13
Unit Test	13
Integration Testing	14
System Testing	14
a) End to End Testing	14
b) Black Box Testing	14
c) Smoke Testing	15
d) Sanity Testing	15
e) Happy path Testing	15
f) Monkey Testing	15

Acceptance Testing	16
a) Alpha Testing	16
b) Beta Testing	16
c) Operational acceptance testing (OAT)	17
Performance Testing	17
a) Load testing	18
b) Stress Testing	18
c) Scalability Testing	18
d) Volume testing (flood testing)	19
e) Endurance Testing (Soak Testing)	19
Other Type of Testing	19
Cross browser testing	19
Browser Compatibility Testing	19
Negative Testing	20
Regression Testing	20
Risk-Based Testing (RBT)	20
Static Testing	21
Bug Report	21
Bug report format	22
Test Cases	23
Why test cases are important	24
Test Case Format	24
API Testing	26
Introduction To API Testing	26
API Testing Methods	26
Response Codes	26
GitHub	30

Introduction To Git	30
Introduction To GitHub	30
GitHub Repository	30

Introduction To Software Testing

Software testing is a method to check whether the software acquires given requirements or not. It also ensures there is no defect in software. It involves execution of software/system components using manual or automated tools. The purpose of software testing is to detect any errors or gaps or missing components. Software testing is one of the important components of a software development life cycle.

Origin of Software Testing

Software testing didn't evolve in a single day; it took time and sweat to get it where it is today. It all started earlier with the programming and debugging phase, when finding errors during debugging was considered testing. In 1957, testing got an individual identity and was treated as a separate activity from the debugging process. Till the late '70s, testing was seen as an exercise to ensure that the software works as per the specified requirements. It was then extended to find the errors, besides ensuring the proper functioning of the software. In the '80s, the testing activity was also considered as a measurement of quality. With this, it gained more importance and was treated as a clearly defined and managed process of the software development life cycle. By the mid-'90s, the testing process had its own life cycle.

Objectives of Testing

There are many objectives of software testing. Some of them are given below:

- **Finding defects:** During the development period, Programmer may create some defect in software, later during software testing we have to find that defect and give it to the programmer for correction.
- **Improve Quality:** Software testing makes sure that software is error free, runs without any problem, which improves the quality of software.
- **Happy Client:** This phase makes sure the requirement given by the client meets or exceeds by the product, which makes the client happy.
- **Company reputation:** Quality of software makes the reputation of a software company high. With the quality of software, error free, without any defect makes the reputation of a software company high.

Requirements for testing

Most software bugs can be traced back to the requirement phase. That's why the best way to decrease the number of new open bugs in a project is to introduce a requirements analysis stage that your teams must complete before they start coding. Here are some requirements for software testing:

- **Completeness:** A requirement must contain all information needed for developers and everyone else who uses it to do their jobs. And a requirement should include all the details necessary to express the users' needs. Before Testing, QA should have complete knowledge about software requirements given by the client.
- **Clearness:** Requirements should be transparent and clear for everyone, with only one interpretation possible. There should not be any confusion. Most importantly QA should have clear ideas about software.
- **Correctness:** Testing a system against incorrect requirements is a waste of time, money, and effort. How correct is your requirement? Is this really what's required from the system, or did someone make a mistake when writing the requirements?
- **Testability:** There should be a way to test the product and check whether the finished product meets the requirement given by the client. Testable products are easy and less costly to maintain, and the chances of achieving customer satisfaction are much higher. That's why testability is so important to the maintainability of applications

Functional and Non-Functional requirements

A functional requirement defines a system or its components. It describes the functions a software must perform. A function is nothing but inputs, behavior and output. Why we need software is defined in functional requirements. It is the core work of a system.

A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?

A functional requirement defines a system or its component whereas a non-functional requirement defines the performance attribute of a software system.

Advantages and Disadvantages of testing

Here are some advantages and disadvantage of software testing:

Advantages:

- **Fast:** As manual testing consumes a great deal of time in both the process of software development as well as during the software application testing, automated tools are a faster option as long as the scripts which need to be done are standard and noncomplex.
- **Reliability:** Automation of test script execution eliminates the possibility of human error when the same sequence of actions is repeated again and again.
- **Reusability:** The test cases can be used in various versions of the software. Not only will project management, client be very grateful for the reduced project time and cost, but it will certainly help us when estimating project costs.

Disadvantages:

- **Hard:** Writing test automation scripts is not easy. You need experienced and knowledgeable QA to write those scripts.
- **Automation Script Errors:** If an error is made in the test automation scripts which is undetected, It could be fatal for the project since the correct testing won't be done.
- **Complexity:** With the increased number of requirements that are to be tested, this leads to more and more complexity which makes the maintenance of test data extremely difficult.

Characteristics of Testing

The followings are the characteristics of testing:

- **Detecting Errors:** To detect maximum errors, the tester should understand the software thoroughly and try to find the possible ways in which the software can fail.

- **No redundancy:** Every test should have a different purpose. There is no need to develop 4 different tests to intend for one outcome. There should be different tests for distinct purposes.
- **Choose the most appropriate test:** To detect maximum error, we should choose carefully, which test is appropriate for that particular scenario.
- **Moderate test:** A good test is considered when it's not too simple nor too complex. It should be between the two. All tests should be performed separately.

Difference between Testing and Debugging

Basically, testing is finding errors and bugs while debugging is eliminating those errors by correcting them. The difference between testing and debugging are given below:

Testing	Debugging
<p>Testing is the process of finding errors and bugs.</p> <ul style="list-style-type: none"> ● It is the process to identify the failure of implemented code. ● Testing is the display of errors. ● Testing can be done by an insider as well as an outsider. ● Testing can be manual or automated. ● It is done by the QA or testers. 	<ul style="list-style-type: none"> ● Debugging is the process of correcting the bugs and errors found while testing. ● It is the process to give the absolution to code failure. ● Debugging is a deductive process. ● Debugging is done only by insiders. Outsiders can't do debugging. ● Debugging is always manual. Debugging can't be automated. ● It is done by a programmer or developer.

Quality Assurance, Quality Control, and Testing

Quality Assurance is focused on planning, documenting and agreeing on a set of guidelines that are necessary to assure quality. QA planning is undertaken at the beginning of a project, and draws on both software specifications and industry or company standards. The purpose of QA is to prevent defects from entering into the solution in the first place.

Quality control includes all activities that are designed to determine the level of quality of the delivered ICT solutions. QC is a reactive means by which quality is gauged and monitored, and QC includes all operational techniques and activities used to fulfill requirements for quality.

Testing is one of the ways of detecting those defects. It is a process to detect errors and bugs. It involves execution of software/system components using manual or automated tools. The purpose of software testing is to detect any errors or gaps or missing components. Software testing is one of the important components of a software development life cycle.

Verification and Validation

Validation is the process of evaluating the final product to check whether the software meets the client's needs. In validation we check whether software functions are correct or not. It is validation of actual and expected product. Validation is dynamic testing. Verification is the process where we check whether we are on the right track or not to make the final product. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing. Verification means Are we building the product, right?

Errors, Defects, and Failures

An error is a mistake or misunderstanding on the developer's side. Generally, when developers enter a wrong loop or syntax then error occurs.

Defects can be defined as variance between expected result and outcome. When our expected result does not match the final result, then there is a defect.

Failures can be defined as inability to perform its function when it reaches the end user.

Failures are observed by testers.

STLC VS SDLC

SDLC

SDLC is basically a software development life cycle where we plan, design, develop and test high-quality software. This life cycle makes the software quality better and it helps to develop software easily. With the proper planning and scheduling this cycle makes software quality improve on time and deliver it to clients. The main aim of SDLC is to provide high quality products that meet the client's requirement with the minimum time and cost.

Phases of SDLC

SDLC contains mainly seven stages i.e., planning, requirement analysis, designing, coding, testing, implementation and maintenance.

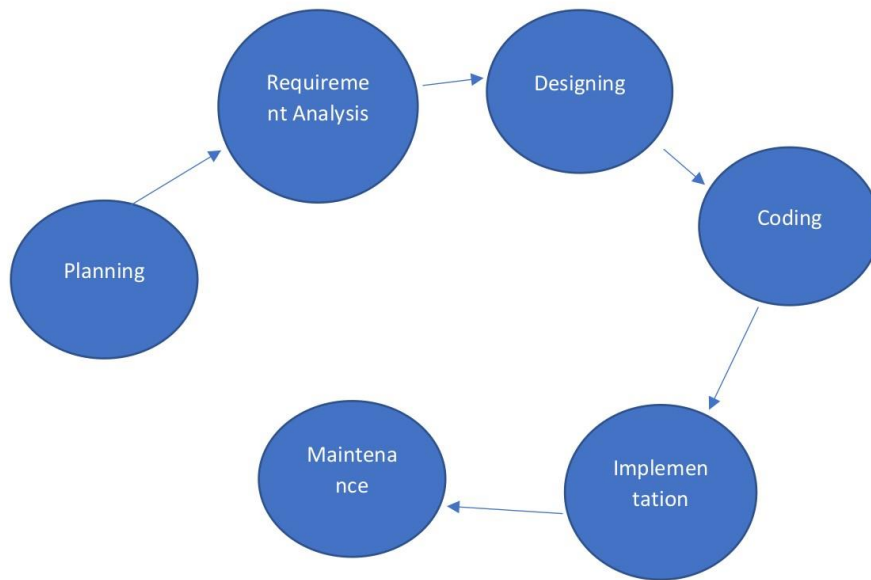


Fig: Software development life cycle

1. **Planning:** In this phase, developers plan for upcoming projects. It helps to define the problem and scope of any existing systems, as well as determine the objectives for their new systems. By developing an effective outline for the

upcoming development cycle, they'll theoretically catch problems before they affect development.

2. **Requirement Analysis:** In this phase the developer will analyze the requirement given by the client. Furthermore they will create software requirement specifications or SRS documents. This includes all the specifications for software, hardware, and network requirements for the system they plan to build.
3. **Design:** Developers will first outline the details for the overall application, such as
 - User interface
 - System Interfaces
 - Network and network requirement
 - Databases

They'll typically turn the SRS document they created into a more logical structure that can later be implemented in a programming language.

4. **Coding:** This stage is the part where developers actually write code and build the application according to the earlier design documents and outlined specifications.

Programming languages can include staples such as C++, PHP, and more. Developers will choose the right programming code to use based on the project specifications and requirements.

5. **Testing:** Building software is not the end. Now it must be tested to make sure that there aren't any bugs and that the end-user experience will not negatively be affected at any point. During the testing stage, developers will go over their software with a fine-tooth comb, noting any bugs or defects that need to be tracked, fixed, and later retested.
6. **Implementation:** After testing, the overall design for the software will come together. Different modules or designs will be integrated into the primary source code through developer efforts, usually by leveraging training environments to detect further errors or defects. The information system will be integrated into its environment and eventually installed.

7. **Maintenance:** Developers are responsible for implementing any changes that the software might need after deployment. This can include handling residual bugs that were not able to be patched before launch or resolving new issues that crop up due to user reports. Larger systems may require longer maintenance stages compared to smaller systems.

STLC

Software testing life cycle (STLC) is a series of activities conducted during the testing process to ensure quality goals are met. It involves both verification and validation activities. Software testing is not just a single activity i.e., testing but it is a series of activities carried out to certify the product quality.

Phases of STLC

There are following six major phases in every software testing life cycle model:

1. **Requirement Analysis:** In this phase the test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with clients to understand requirements in detail. Main activities in this phase are given below:
 - Identify types of tests to be performed.
 - Gather details about testing priorities and focus.
 - Prepare requirement traceability matrix (RTM).
 - Identify test environment details where testing is supposed to be carried out.
 - Automation feasibility analysis.
2. **Test Planning:** In this phase senior QA determines the test plan strategy. Also in this phase , the resource, test environment, test limitations and the testing schedule are also determined. The main activities in this phase are given below:
 - Preparation of test plan/strategy document for various types of testing
 - Test tool selection
 - Test effort estimation
 - Resource planning and determining roles and responsibilities.
 - Training requirement

3. **Test case development:** In this phase, involves the creation, verification and rework of test cases & scripts after the test plan is ready. The QA team starts the development process of test cases for individual units. The main activities in this phase are given below:
 - Create test cases, automation scripts
 - Review and baseline test cases and scripts
 - Create test data
4. **Test environment setup:** This phase decides the software and hardware conditions under which a work product is tested. It can be done in parallel with the test case development phase. Test team may not be involved in this activity if the development team provides the test environment. The main activities in this phase are given below:
 - Understand the required architecture, environment set-up and prepare hardware and software requirement list for the test environment.
 - Setup test environment and test data. ● Perform smoke tests on the build.
5. **Test execution:** This phase is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test scripts execution, test script maintenance and bug reporting. If bugs are reported then it goes back to the development team for correction and retesting will be performed. The main activities in this phase are given below:
 - Execute tests as per plan
 - Document test results, and log defects for failed cases
 - Map defects to test cases in RTM
 - Retest the defect fixes
 - Track the defects to closure
6. **Test cycle closure:** This phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing teams discuss and analyze testing artifacts to identify strategies that have to be implemented in future. The main activities in this phase are given below:
 - Evaluate cycle completion
 - Prepare test metrics based on the above parameters.
 - Document the learning out of the project
 - Prepare test closure report

Difference between SDLC and STLC

SDLC	STLC
<ul style="list-style-type: none">● SDLC is mainly related to software development.● Other than development, it also includes software testing.● In SDLC, more number of member (Developer) required.● In SDLC, the development team makes plans and designs.● It helps in developing quality software.	<ul style="list-style-type: none">● STLC is mainly related to software testing.● STLC only focuses on software testing.● In STLC, less number of members (Tester) required. In● STLC, the tester team makes plans and design.● It helps in making the software defect free.

Types of Testing

Unit Test

Unit testing is a type of software testing which is done on an individual unit or component to test its corrections. Typically, Unit testing is done by the developer at the application development phase. Each unit in unit testing can be viewed as a method, function, procedure, or object. Developers often use test automation tools such as NUnit, Xunit, JUnit for the test execution.

Unit testing is important because we can find more defects at the unit test level.

For example, there is a simple calculator application. The developer can write the unit test to check if the user can enter two numbers and get the correct sum for additional functionality.

Integration Testing

Integration testing is a type of software testing where two or more modules of an application are logically grouped together and tested as a whole. The focus of this type of testing is to find the defect on interface, communication, and data flow among modules. A Top-down or Bottom-up approach is used while integrating modules into the whole system.

This type of testing is done on integrating modules of a system or between systems. **For example**, a user is buying a flight ticket from any airline website. Users can see flight details and payment information while buying a ticket, but flight details and payment processing are two different systems. Integration testing should be done while integrating the airline website and payment processing system.

System Testing

System testing is a type of testing where the tester evaluates the whole system against the specified requirements.

a) End to End Testing

It involves testing a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

For example, a tester is testing a pet insurance website. End to End testing involves testing of buying an insurance policy, LPM, tag, adding another pet, updating credit card information on users' accounts, updating user address information, receiving order confirmation emails and policy documents.

b) Black Box Testing

Blackbox testing is a software testing technique in which testing is performed without knowing the internal structure, design, or code of a system under test. Testers should focus only on the input and output of test objects.

c) Smoke Testing

Smoke testing is performed to verify that basic and critical functionality of the system under test is working fine at a very high level.

Whenever a new build is provided by the development team, then the Software Testing team validates the build and ensures that no major issue exists. The testing team will ensure that the build is stable, and a detailed level of testing will be carried out further.

For example, a tester is testing a pet insurance website. Buying an insurance policy, adding another pet, providing quotes are all basic and critical functionality of the application. Smoke testing for this website verifies that all these functionalities are working fine before doing any in-depth testing.

d) Sanity Testing

Sanity testing is performed on a system to verify that newly added functionality or bug fixes are working fine. Sanity testing is done on stable builds. It is a subset of the regression test.

For example, a tester is testing a pet insurance website. There is a change in the discount for buying a policy for a second pet. Then sanity testing is only performed on buying insurance policy modules.

e) Happy path Testing

The objective of Happy Path Testing is to test an application successfully on a positive flow. It does not look for negative or error conditions. The focus is only on valid and positive inputs through which the application generates the expected output.

f) Monkey Testing

Monkey Testing is carried out by a tester, assuming that if the monkey uses the application, then random input and values will be entered by the Monkey without any knowledge or understanding of the application.

The objective of Monkey Testing is to check if an application or system gets crashed by providing random input values/data. Monkey Testing is performed randomly, no test cases are scripted, and it is not necessary to be aware of

the full functionality of the system.

Acceptance Testing

Acceptance testing is a type of testing where client/business/customer test the software with real time business scenarios.

The client accepts the software only when all the features and functionalities work as expected. This is the last phase of testing, after which the software goes into production. This is also called User Acceptance Testing (UAT).

a) Alpha Testing

Alpha testing is a type of acceptance testing performed by the team in an organization to find as many defects as possible before releasing software to customers.

For example, the pet insurance website is under UAT. The UAT team will run real-time scenarios like buying an insurance policy, buying annual membership, changing the address, ownership transfer of the pet in the same way the user uses the real website. The team can use test credit card information to process payment-related scenarios.

b) Beta Testing

Beta Testing is a type of software testing which is carried out by the clients/customers. It is performed in the **Real Environment** before releasing the product to the market for the actual end-users.

Beta Testing is carried out to ensure that there are no major failures in the software or product, and it satisfies the business requirements from an end-user perspective. Beta Testing is successful when the customer accepts the software.

Usually, this testing is typically done by the end-users. This is the final testing done before releasing the application for commercial purposes. Usually, the Beta version of the software or product released is limited to a certain number of users in a specific area.

So, the end-user uses the software and shares the feedback with the company. The company then takes necessary action before releasing the software worldwide.

c) Operational acceptance testing (OAT)

Operational acceptance testing of the system is performed by operations or system administration staff in the production environment. The purpose of operational acceptance testing is to make sure that the system administrators can keep the system working properly for the users in a real-time environment.

The focus of the OAT is on the following points:

- Testing of backup and restore.
- Installing, uninstalling, upgrading software.
- The recovery process in case of natural disaster.
- User management.
- Maintenance of the software.

Performance Testing

Performance testing is testing of an application's stability and response time by applying load.

The word stability means the ability of the application to withstand the presence of load. Response time is how quickly an application is available to users. Performance testing is done with the help of tools. Loader.IO, JMeter, LoadRunner, etc. are good tools available in the market.

a) Load testing

Load testing is testing of an application's stability and response time by applying load, which is equal to or less than the designed number of users for an application.

For example, your application handles 100 users at a time with a response time of 3 seconds, then load testing can be done by applying a load of the maximum of 100 or less than 100 users. The goal is to verify that the application is responding within 3 seconds for all the users.

b) Stress Testing

Stress testing is testing an application's stability and response time by applying load, which is more than the designed number of users for an application.

For example, your application handles 1000 users at a time with a response time of 4 seconds, then stress testing can be done by applying a load of more than 1000 users. Test the application with 1100,1200,1300 users and notice the response time. The goal is to verify the stability of an application under stress.

c) Scalability Testing

Scalability testing is testing an application's stability and response time by applying load, which is more than the designed number of users for an application.

For example, your application handles 1000 users at a time with a response time of 2 seconds, then scalability testing can be done by applying a load of more than 1000 users and gradually increasing the number of users to find out where exactly my application is crashing.

Let's say my application is giving response time as follows:

- 1000 users -2 sec
- 1400 users -2 sec
- 4000 users -3 sec
- 5000 users -45 sec

- 5150 users- crash – This is the point that needs to identify in scalability testing

d) Volume testing (flood testing)

Volume testing is testing an application's stability and response time by transferring a large volume of data to the database. Basically, it tests the capacity of the database to handle the data.

e) Endurance Testing (Soak Testing)

Endurance testing is testing an application's stability and response time by applying load continuously for a longer period to verify that the application is working fine.

For example, car companies soak testing to verify that users can drive cars continuously for hours without any problem.

Other Type of Testing

Cross browser testing

Cross browser testing is testing an application on different browsers, operating systems, mobile devices to see look and feel and performance.

Why do we need cross-browser testing? The answer is different users use different operating systems, different browsers, and different mobile devices. The goal of the company is to get a good user experience regardless of those devices.

Browser stack provides all the versions of all the browsers and all mobile devices to test the application. For learning purposes, it is good to take the free trial given by browser stack for a few days.

Browser Compatibility Testing

This is a sub-type of Compatibility Testing (which is explained below) and is performed by the testing team.

Browser Compatibility Testing is performed for web applications and ensures that the software can run with a combination of different browsers and operating systems. This type of testing also validates whether a web application runs on all versions of all browsers or not.

Negative Testing

The mindset of the tester is to “Break the System/Application” and it is achieved through Negative Testing.

Negative Testing technique is performed using incorrect data, invalid data, or input. It validates if the system throws an error of invalid input and behaves as expected.

It should not take much time to load any page or system and should be sustained during peak load. Different performance and load tools are used to do this testing.

Regression Testing

Regression testing is testing of unchanged features of the application to make sure that any bug fixes, adding new features, deleting, or updating existing features, are not impacting the working application.

To find out regression scope is an important part in Regression Testing. To find out the regression scope, Tester needs to find out the area of application where changes happened and the Impact of those changes on the entire application.

It is difficult to cover the whole regression test suite in every release, so Automation Testing Tools are used in regression testing.

Risk-Based Testing (RBT)

For Risk-Based Testing, the functionalities or requirements are tested based on their priority. Risk-Based Testing includes testing of highly critical functionality, which has the highest impact on business and in which the probability of failure is very high.

Priority decisions are based on business needs, so once priority is set for all functionalities, then high priority functionality or test cases are executed first, followed by medium and then low priority functionalities.

Low priority functionality may be tested or not tested based on the available time. Risk-Based Testing is carried out if there is insufficient time available to test the entire software and the software needs to be implemented on time without any delay.

This approach is followed only by the discussion and approval of the client and senior management of the organization.

Static Testing

Static Testing is a type of testing which is done without the execution of any code. Reviews, walkthroughs, and inspections are different methods of performing static testing. Activities like reviewing of requirement documents, customer requirement specification, high level, and low-level design, code syntax, naming standards, etc. come under static testing.

Static Testing also applies to test cases, test plans, test scenarios. Static testing is done to prevent the defect rather than catching the defect at a later stage. That is why static testing is cost-effective.

For example, Tester is testing a pet insurance website. The logic for premium calculation is described in requirement documentation. As a part of static testing, testers can review the developer code for premium calculation and compare it with the requirement document to prevent the defect related to premium calculation.

Bug Report

A software bug report contains specific information about what is wrong and what needs to be fixed in software or on a website. These comprehensive reports include requests or specifics for each software issue. It lets the developers understand what is wrong and how to fix it.

Now that we have established that bugs aren't good, and what bug reports are, let us discuss why we need them.

A clear and concise software bug report can help developers quickly understand what problems a web application or software is experiencing. The clear communication of issues with these reports will lead to fixing the bugs as soon as possible to provide users with a seamless experience.

Bug report format

Depending on the bug tracking system, your template could differ. However, the following fields are required:

- Title
- Severity/Priority
- Description
- Steps to reproduce
- Expected result
- Actual result
- Attachments (screenshots, videos, text)
- Status
- Developer Remarks

Here is sample of bug report:


S.N	Bug Image/Description	Status	Dev Remarks
1	<p>Title:Role Management</p> <p>Steps to be reproduced: 1.Click on Role Management</p> <p>Expected Result: Serial Number should be shown as Order.</p> <p>Actual Result: Serial number is showing in wrong order</p> 	Open	

Fig: Sample Bug Report

Test Cases

A test case is a set of actions performed on a system to determine if it satisfies software requirements and functions correctly. The purpose of a test case is to determine if different features within a system are performing as expected and to confirm that the system satisfies all related standards, guidelines and customer requirements. The process of writing a test case can also help reveal errors or defects within the system.

Test cases are typically written by members of the quality assurance (QA) team or the testing team and can be used as step-by-step instructions for each system test.

Testing begins once the development team has finished a system feature or set of features. A sequence or collection of test cases is called a test suite.

A test case document includes test steps, test data, preconditions and the post conditions that verify requirements.

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.

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.

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 Case Format

Test cases must be designed to fully reflect the software application features and functionality under evaluation. QA engineers should write test cases so only one thing is tested at a time. The language used to write a test case should be simple and easy to understand, active instead of passive, and exact and consistent when naming elements.

The components of a test case include:

- **Test name.** A title that describes the functionality or feature that the test is verifying.
- **Test ID.** Typically a numeric or alphanumeric identifier that QA engineers and testers use to group test cases into test suites.
- **Test Type.** This typically indicates whether it is functional or UI.
- **Test Description** This important component describes what the test intends to verify in one to two sentences.
- **Test Data.** The data required to test in that module are written inside test data.
- **Expected Output.** An outline of how the system should respond to each test step.
- **Test steps.** Detailed descriptions of the sequential actions that must be taken to complete the test.
- **Actual results.** After executing the test we write the result in Actual Result.
- **Pass/Fail.** If actual result and expected result is matched then we write pass otherwise we write fail in this column. Here is the example of test case

S.No.	Test Case Id	Type of Test Case	Test Case Description	Steps to be Executed	Test Data	Expected Output	Actual Output	Pass/Fail
1	LOGIN_01	UI	Verify login UI	1. Open the Browser 2. Check for the finalized design/UI		No Issue in Design	As Expected	Pass
2	LOGIN_02		Verify login with valid credentials	1. Open the Browser 2. Enter the Web Portal Link 3. Enter Username 4. Enter Password 5. Click Login	Username: 9851045150 Password: 123456789	User should be logged in successfully.	As Expected	Pass
3	LOGIN_03		Verify login with invalid credentials	1. Open the Browser 2. Enter the Web Portal Link 3. Enter Valid Username 4. Enter Invalid Password 5. Click Login	Username: Superadmin Password: 123456	- User should not be logged in. - Should throw message saying Invalid Password	As Expected	Pass
4	LOGIN_04			1. Open the Browser 2. Enter the Web Portal Link. 3. Enter Invalid Username 4. Enter Valid Password 5. Click Login	Username: 9876543210 Password: superuser	- User should not be logged in. - Should throw message saying Invalid Username	As Expected	Pass

Fig: Test Case Sample

API Testing

Introduction To API Testing

API Testing is an open-source web automation testing technique that is used for testing RESTful APIs for web applications. The purpose of rest api testing is to record the response of rest api by sending various HTTP/S requests to check if rest api is working fine or not. Rest api testing is done by GET, POST, PUT and DELETE methods.

Rest stands for Representational State Transfer. It is an architectural style and an approach for communication used in the development of Web Services. REST has become a logical choice for building APIs. It enables users to connect and interact with cloud services efficiently.

An API or Application Programming Interface is a set of programming instructions for accessing a web-based software application.

In other words, a set of commands used by an individual program to communicate with one another directly and use each other's functions to get information.

API Testing Methods

There are mainly 4 types of API Testing methods: GET, POST, Delete, and PUT.

- **GET**– The GET method is used to extract information from the given server using a given URI. While using GET requests, it should only extract data and should have no other effect on the data.
- **POST**– A POST request is used to create a new entity. It can also be used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
- **PUT**– Create a new entity or update an existing one.
- **DELETE**– Removes all current representations of the target resource given by a URI.

Response Codes

Here are some of Important response codes:

HTTP status codes

100's (Informational) : Communicates transfer-protocol level information.

200's (Success) : This line of codes indicates that the client's request was accepted successfully.

300's (Redirection) : This line of code indicates that the client must take some additional action in order to complete their request.

400's (Client Error) : This line of code indicates that there is an error from the client side.

500's (Server Error) : This line of code indicates that there is an error from the server side.

STATUS CODE	MEANING
1xx Information	
100	Continue
101	Switching Protocols
102	Processing
103	Early Hints
2xx Successful	
200	Ok
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Rest Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used
3xx Redirection	
300	Multiple Choices
301	Moved permanently
302	Found (Previously "Moved Temporarily")
303	See Other

304	Not Modified
305	Use Proxy
306	Switch Proxy
307	Temporary Redirect
308	Permanent Redirect
4xx Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	URI Too Long
415	Unsupported Media Type
416	Range Not Satisfiable
417	Expectation Failed
418	I'm a Teapot
421	Misdirected Request
422	UnProcessable Entity
423	Locked
424	Failed Dependency
425	Too Early
426	Upgrade Required

428	Precondition Required
429	Too Many Requests
431	Request Header fields Too Large
451	Unavailable For Legal Reasons
5xx Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Loop Detected
511	Network Authentication Required

GitHub

Introduction To Git

Git is one of the most popular version control systems. It is one of the distributed version control systems. It is an open-source project which is compatible with many operating systems and IDEs.

It allows us to track changes in an application, or in a folder, or in a single file over time across different users, and different computers. Git takes a snapshot of all files whenever we create a new commit. Every committer on a project always has a copy of the whole repository on their machine. So, we can commit to the Git file system also in offline mode.

Introduction To GitHub

GitHub is a cloud-based Git repository hosting service. It lets individuals and teams work together on projects. Git is a command-line tool whereas GitHub comes with a web-based graphical interface. GitHub is an application allowing you to store remote repositories on their servers. It also provides a user-friendly platform to interact with and manage your repositories. It is a public platform that allows millions of users to share their projects with the world.

The main purpose is to allow people to collaborate together to build projects. But it's not just limited to collaboration. On top of that, it can be used as a portfolio for your best work. One of the most useful features of GitHub is being able to access your repository from any location. Also, it's an industry standard for hosting Git repositories.

GitHub Repository

Repositories, usually called '**repos**', store the full history and source control of a project. They can either be hosted locally or on a shared server such as GitHub. Most repositories are stored on GitHub, while core contributors make copies of the repository on their machine and update the repository using the push/pull system. Any repository stored somewhere other than locally is called a 'remote repository'.

First of all, pull all the information from your remote repository and make a direct copy of it in your local machine.

The working directory is your project at its current state and time. It contains all your files and everything at its current state. Let's say you made a new file (test.py) and wrote some code in this file. So now you want to change or take this and take a snapshot of it and put it in your source control.

So after making the changes, you add your file to the staging area and commit your changes. Commit is just pretty much taking a snapshot and putting it in source code. The staging area is the bundle of all the modifications to the project that are going to be committed. So after you add your file to the staging area and commit, you then put them in the local repository.

So, you have made a bunch of changes and you have changed the local repository. Now local and remote repositories don't match. Now you would push it to the remote repository and now they both matched up.

Git Cheat Sheet

Setup

Set the name and email that will be attached to your commits and tags

```
$ git config --global
user.name "Danny Adams"
$ git config --global
user.email "my-
email@gmail.com"
```

Start a Project

Create a local repo (omit <directory> to initialise the current directory as a git repo)

```
$ git init <directory>
```

Download a remote repo

```
$ git clone <url>
```

Make a Change

Add a file to staging

```
$ git add <file>
```

Stage all files

```
$ git add .
```

Commit all staged files to git

```
$ git commit -m "commit
message"
```

Add all changes made to tracked files & commit

```
$ git commit -am "commit
message"
```

Basic Concepts

main: default development branch

origin: default upstream repo

HEAD: current branch

HEAD^: parent of HEAD

HEAD~4: great-great grandparent of HEAD

By @DoableDanny

Branches

List all local branches. Add -r flag to show all remote branches. -a flag for all branches.

```
$ git branch
```

Create a new branch

```
$ git branch <new-branch>
```

Switch to a branch & update the working directory

```
$ git checkout <branch>
```

Create a new branch and switch to it

```
$ git checkout -b <new-
branch>
```

Delete a merged branch

```
$ git branch -d <branch>
```

Delete a branch, whether merged or not

```
$ git branch -D <branch>
```

Add a tag to current commit (often used for new version releases)

```
$ git tag <tag-name>
```

Merging

Merge branch a into branch b. Add --no-ff option for no-fast-forward merge



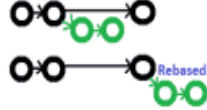
```
$ git checkout b
$ git merge a
```

Merge & squash all commits into one new commit

```
$ git merge --squash a
```

Rebasing

Rebase feature branch onto main (to incorporate new changes made to main). Prevents unnecessary merge commits into feature, keeping history clean



```
$ git checkout feature
$ git rebase main
```

Iteratively clean up a branches commits before rebasing onto main

```
$ git rebase -i main
```

Iteratively rebase the last 3 commits on current branch

```
$ git rebase -i Head~3
```

Undoing Things

Move (&/or rename) a file & stage move

```
$ git mv <existing_path>
<new_path>
```

Remove a file from working directory & staging area, then stage the removal

```
$ git rm <file>
```

Remove from staging area only

```
$ git rm --cached <file>
```

View a previous commit (READ only)

```
$ git checkout <commit_ID>
```

Create a new commit, reverting the changes from a specified commit

```
$ git revert <commit_ID>
```

Go back to a previous commit & delete all commits ahead of it (revert is safer). Add --hard flag to also delete workspace changes (BE VERY CAREFUL)

```
$ git reset <commit_ID>
```

Review your Repo

List new or modified files not yet committed

```
$ git status
```

List commit history, with respective IDs

```
$ git log --oneline
```

Show changes to unstaged files. For changes to staged files, add --cached option

```
$ git diff
```

Show changes between two commits

```
$ git diff commit1_ID
commit2_ID
```

Stashing

Store modified & staged changes. To include untracked files, add -u flag. For untracked & ignored files, add -a flag.

```
$ git stash
```

As above, but add a comment.

```
$ git stash save "comment"
```

Partial stash. Stash just a single file, a collection of files, or individual changes from within files

```
$ git stash -p
```

List all stashes

```
$ git stash list
```

Re-apply the stash without deleting it

```
$ git stash apply
```

Re-apply the stash at index 2, then delete it from the stash list. Omit stash@{n} to pop the most recent stash.

```
$ git stash pop stash@{2}
```

Show the diff summary of stash 1. Pass the -p flag to see the full diff.

```
$ git stash show stash@{1}
```

Delete stash at index 1. Omit stash@{n} to delete last stash made

```
$ git stash drop stash@{1}
```

Delete all stashes

```
$ git stash clear
```

Synchronizing

Add a remote repo

```
$ git remote add <alias>
<url>
```

View all remote connections. Add -v flag to view urls.

```
$ git remote
```

Remove a connection

```
$ git remote remove <alias>
```

Rename a connection

```
$ git remote rename <old>
<new>
```

Fetch all branches from remote repo (no merge)

```
$ git fetch <alias>
```

Fetch a specific branch

```
$ git fetch <alias> <branch>
```

Fetch the remote repo's copy of the current branch, then merge

```
$ git pull
```

Move (rebase) your local changes onto the top of new changes made to the remote repo (for clean, linear history)

```
$ git pull --rebase <alias>
```

Upload local content to remote repo

```
$ git push <alias>
```

Upload to a branch (can then pull request)

```
$ git push <alias> <branch>
```