What is Software Testing?

Human beings are prone to mistakes because of in-attention, incorrect assumptions, carelessness, or inadequate system knowledge. This very nature of humans makes software vulnerable to bugs, defects, and errors (we will get to know these terms in detail in later posts). To prevent and correct these issues, we need testing.

Traditionally software testing was done in a single phase, and that too only once the implementation or coding used to get completed. But the increasing complexity of software applications led to the evolution of testing.

So testing techniques have evolved over time, and testing activities are not confined to a single phase. Instead, these were integrated with the different phases of the software development life cycle.
In this blog post, we will walk you through the nitty-gritty of software testing.

What is Software Testing?

Software testing is the process of evaluating a system with the intent of finding bugs. It is performed to check if the system satisfies its specified requirements and quality standards. It evaluates the system to validate its functionality.

Why is Software Testing Important?

Software Testing is an indispensable activity in SDLC because it uncovers any bugs or errors early that can be addressed before releasing the system to the market.

The following are some significant reasons stating the importance of software testing:

- It provides an assurance to the stakeholders that the product works as intended.
- Delivering an untested product with avoidable defects to the end-user/customer leaves a bad reputation for the development company.
- The separate testing phase adds a confidence factor to the stakeholders regarding the quality of the software under development.
- Defects detected in the earlier phase of SDLC result in lesser cost and resource utilization for defect resolution.
- The testing team adds another dimension to software development by providing a different viewpoint to the product development process.
- An untested software not only makes software error-prone but also costs the customer business failure, like in the case of Microsoft’s MP3 player – Zune’s crash.
-

The following are some real-world examples stating the importance of software testing where bugs or errors proved expensive and life-threatening.

- Starbucks’ POS system had glitches. Hence, 60% of stores in the United States and Canada were closed. For some time, the store even served coffee for free, as customers were not able to perform transactions.
- In 2015, a fighter plane F-35 had bugs that made it unable to detect targets correctly.
- Due to a small software bug, China Airlines Airbus A300 crashed, in which 264 innocents lost their lives.
- The costliest accident in history was a software bug in a military satellite launch. It cost $1.2 billion.
- A software bug at a major US bank resulted in the credit of 920 million US dollars to 823 customers.

-

Who Conducts Software Testing?

Testing is/can be done by all technical and non-technical people associated with software development. As testing is not a single process and a series of phases, the following are people associated with it in different phases:

- **Developer** – Developers perform unit testing of the software and ensure that the individual modules or components work correctly.
- **Testers** – Testers are the face of software testing. A tester verifies the application's functionality as a functional tester, checks its performance as a performance tester, automates the functional test cases, and creates test scripts as an automation tester.
- **Test Managers/Lead/Architects** – They develop and define the test strategy and test plan documents.
- **End Users** – A group of end-users does the User Acceptance Testing (UAT) to ensure the software is ready to release to the real world.
-

How is Software Testing done?

Testing can be done both manually as well as using automation tools.

- **Manual Testing** – When performed manually, it is called Manual Testing. It includes requirements verification, development of test strategy and plan, test case preparation, test case execution, defect creation, defect retesting, and finally, test report sharing with all the relevant stakeholders.

- **Automation Testing** – When automated tools are used, it is called Automation testing. It includes test script preparation and test report generation using different tools like – Selenium, Katalon Studio, QTP, etc.
-

  In this type of testing, rerunning test scenarios is quick and easy. It increases test coverage, improves accuracy, and significantly saves time and money compared to manual testing.

Some other types of software testing include functional and non-functional.

Testing Type	Example
Functional Testing	Unit testing Integration testing System testing Acceptance testing Regression testing
Non-Functional Testing	Performance testing Load testing Stress testing Endurance testing Volume testing Security testing Recovery testing

When do we start Software Testing?

Based on the software project and the selection of a specific SDLC model, the testing activities can be performed in the different phases of the software life cycle.

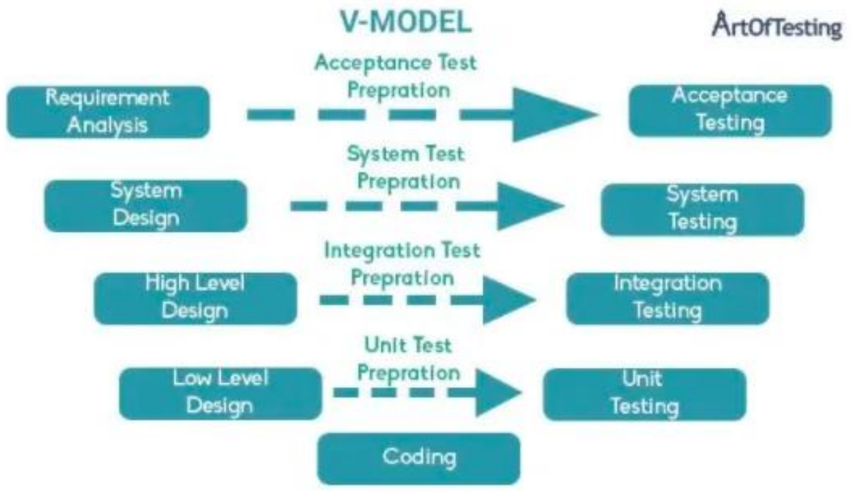
There is a misconception that testing is done only when some part of the software is built. However, **testing can (should)** be started even before a single line of code is written. It can be done in parallel with the development phase, e.g., in the case of the V Model, development and testing activities are integrated.

Development Phase	Testing Activity
Requirement Design	Acceptance test creation
Functional Specification	Functional test case creation
Implementation	Unit test case creation
Code Complete	Test case execution

V Model

What is V Model?

V model is also known as the Verification and Validation model. In the V model, the testing phase goes in parallel with the development phase. Thus, the testing phase starts right at the beginning of SDLC.



As we can see in the above diagram, the test activities start in parallel with the development activities e.g. during the requirement analysis phase, acceptance-test cases are prepared by the testing team; during the system design phase, the system test case is prepared. Similarly, for each phase of development, a corresponding QA activity is performed. Later, when the deliverable gets ready, the QA artifacts are used to conduct the testing. Along with that, each phase of the development phase is verified before moving to the next phase.

Advantages of V Model

- Each phase of development is tested before moving to next phase, hence there is a higher rate of success.
- It avoids defect leakage to the later phases as each phase is verified explicitly.

- The model has clear and defined steps. So, it is easier to implement.
- It is suitable for smaller projects where requirements are fixed.

Disadvantages of V Model

- The testing team starts in parallel with development. Hence, the overall budget and resource usage increases.
- Change in requirement are difficult to incorporate.
- The working model of the software is only available in the later phases of the development.
- It is not suitable for complex and large applications because of its rigid process.

Test Strategy Document

April 29, 2023

Hello friends, in this article, we will study test strategy documents, their templates, and some tips to create a good test strategy document. But before that let's first study – what test strategy is.

What is test strategy?

As the name suggests, test strategy means strategy employed to test a particular software application. In other words, a Test Strategy is an outline or approach to the way testing will be carried out in the software development life cycle. Its purpose is to define the exact process the testing team will follow to achieve the organizational objectives from a testing perspective.

Test strategy approaches

The the different approaches to test strategy that are employed are-

1. **Analytical approach** – This approach is based on risk analysis based on the project's requirements and different stakeholder's input. Based on this risk analysis, a test strategy is developed to plan, design, and prioritize the testing efforts.

2. **Model-based approach** – This approach uses various statistical models for developing a test strategy.
3. **Consultative approach** – In this approach, a test strategy is developed based on consultation with technology or domain experts.
4. **Methodical approach** – This approach is simply based on using a pre-defined set of testing approaches that may relate to a particular type of application testing.
5. **Dynamic or Heuristic approach** – The heuristic approach developed by James Basch is based on the exploratory techniques instead of pre-planned.
6. **Standard-compliant approach** – With this approach the test strategy is prepared based on the industry standards and processes.

What is a test strategy document?

A test strategy document is a **high-level document describing the way testing will be carried out**. In this, we document the test objectives and set of guidelines for achieving those objectives. It is presented by the project manager to all the stakeholders in the testing process. It can have a scope of an entire organization or a particular project.

Test Plan

What is a Test Plan?

A Test Plan is a formal document derived from requirement documents like Software Requirement Specification, Use Case documents, etc. It describes in detail, the scope of testing and the different activities performed in testing.

It is generally prepared by a test manager and approved by the different stakeholders of the application.

Features of a Test Plan

A Test Plan needs to address the following-

- The overall scope of testing
- Risk Analysis
- Test estimate
- Resource Requirement
- Tools used
- Scheduling, review, and analysis of test design activities
- Creation of test cases and test data
- Identification of test monitoring and test control activities

Difference Between Test Plan and Test Strategy

A test plan is a formal document derived from requirement documents. It describes in detail, the scope of testing and the different activities performed during testing.

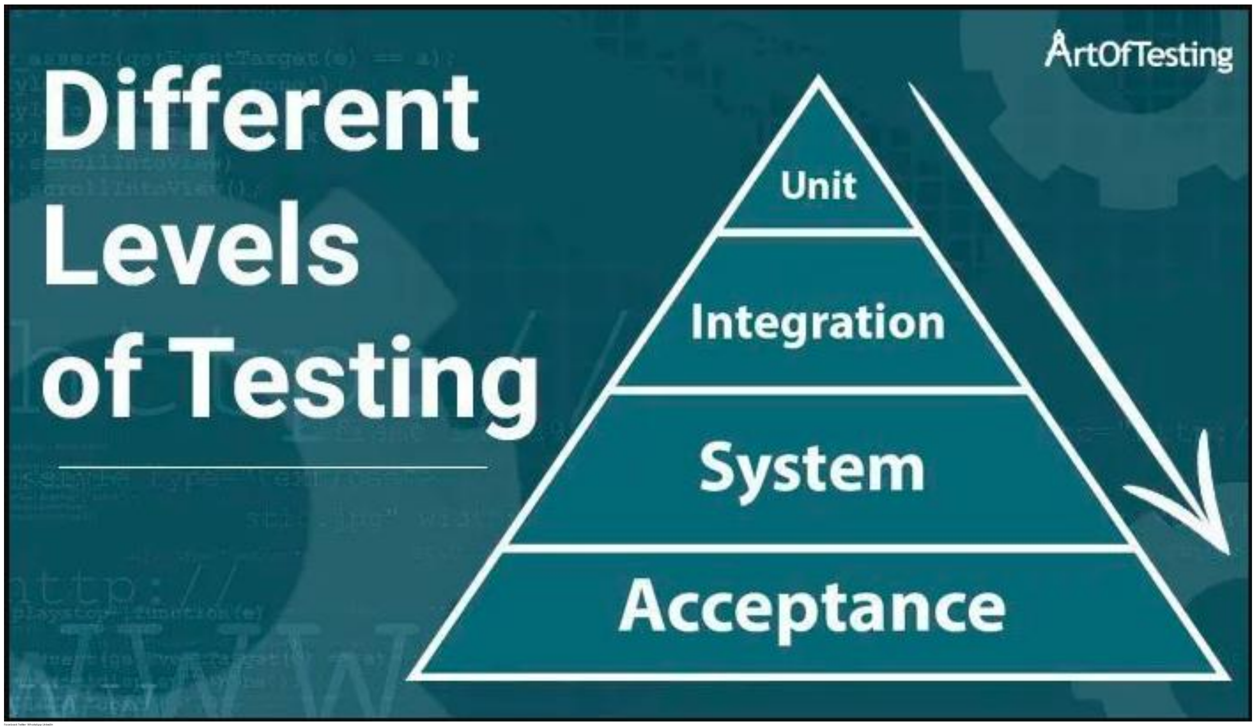
Whereas, a test strategy is a high-level document describing the way testing will be carried out in an organization. It basically aims at providing a systematic approach to the software testing process.

Test Plan vs Test Strategy

Test Plan	**Test Strategy**
A test plan describes in detail the scope of testing and the different activities performed in testing.	A test strategy is a high-level document containing some guidelines about the way testing will be carried out.
A test plan is specific to a particular project.	A test strategy is usually for a complete organization.
It describes the whole testing activities in detail – the techniques used, schedule, resources, etc.	It describes the high-level test design techniques to be used, environment specifications, etc.
It is prepared by the test lead or test manager.	It is generally prepared by the project manager.
A test plan document includes components like – features to be tested, components not to be tested, approach to testing, pass-fail	A test strategy document includes – scope, test approach, testing tools, test environment specifications, release control, risk analysis, etc.

Test Plan	**Test Strategy**
criteria, test deliverables, estimates, assumptions, etc.	
Test plans can be changed or updated.	Test strategy is usually not changed.
It is about the details and specifics.	It is more about general approaches and methodologies.

Levels of Testing



Software testing can be performed at different levels of the software development process. Performing testing activities at multiple levels help in the early identification of bugs and better quality of software product. In this tutorial, we will be studying the different levels of testing namely – Unit Testing, Integration Testing, System Testing, and Acceptance Testing.

Unit Testing

- Unit Testing is the first level of testing usually performed by the developers.
- In unit testing, a module or component is tested in isolation.
- As the testing is limited to a particular module or component, exhaustive testing is possible.
- Advantage – Error can be detected at an early stage saving time and money to fix it.
- Limitation – Integration issues are not detected in this stage, modules may work perfectly on isolation but can have issues in interfacing between the modules.

Integration Testing

- Integration testing is the second level of testing in which we test a group of related modules.
- It aims at finding interfacing issues b/w the modules i.e. if the individual units can be integrated into a sub-system correctly.
- It is of four types – Big-bang, top-down, bottom-up, and Hybrid.
    1. In **big bang integration**, all the modules are first required to be completed and then integrated. After integration, testing is carried out on the integrated unit as a whole.

    2. In **top-down integration** testing, the testing flow starts from top-level modules that are higher in the hierarchy towards the lower-level modules. As there is a possibility that the lower-level modules might not have been developed while beginning with top-level modules.

       So, in those cases, stubs are used which are nothing but dummy modules or functions that simulate the functioning of a module by accepting the parameters received by the module and giving an acceptable result.

    3. **Bottom-up integration testing** is also based on an incremental approach but it starts from lower-level modules, moving upwards to the higher-level modules. Again the higher-level modules might not have been developed by the time lower modules are tested. So, in those cases, drivers are used. These drivers simulate the functionality of higher-level modules in order to test lower-level modules.

    4. **Hybrid integration testing** is also called the Sandwich integration approach. This approach is a combination of both top-down and bottom-up

integration testing. Here, the integration starts from the middle layer, and testing is carried out in both directions, making use of both stubs and drivers, whenever necessary.

System Testing

- System Testing is the third level of testing.
- It is the level of testing where the complete integrated application is tested as a whole.
- It aims at determining if the application conforms to its business requirements.
- System testing is carried out in an environment that is very similar to the production environment.

Acceptance Testing

- Acceptance testing is the final and one of the most important levels of testing on successful completion of which the application is released to production.
- It aims at ensuring that the product meets the specified business requirements within the defined standard of quality.
- There are two kinds of acceptance testing- alpha testing and beta testing.
    1. When acceptance testing is carried out by testers or some other internal employees of the organization at the developer's site it is known as **alpha testing**.
    2. User acceptance testing done by end-users at the end-user's site is called **beta testing**.

This concludes our tutorial on the different levels of testing. You can continue with our Software Testing Tutorial course to study these levels of software testing in detail, in the coming tutorials.

Test Scenario – Definition, Template and Examples

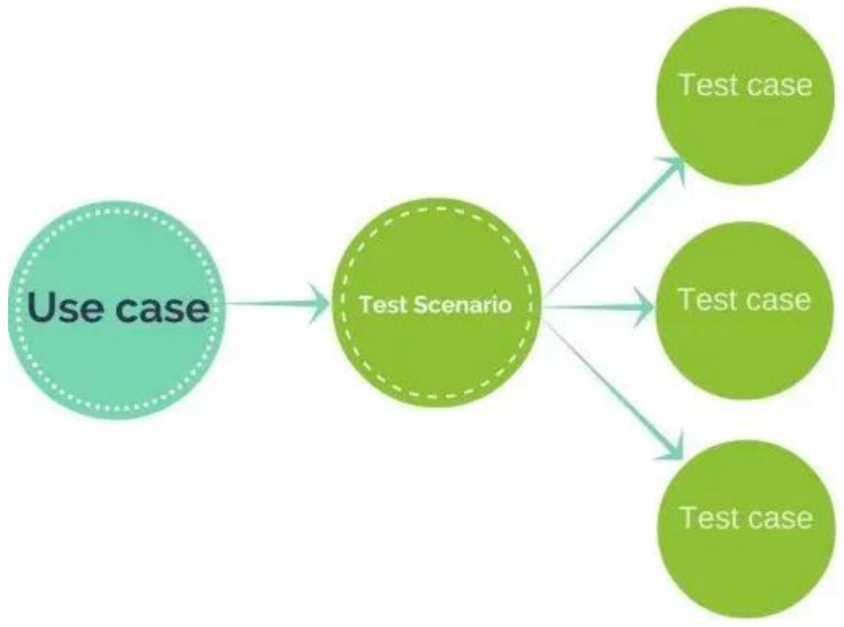
In this tutorial, we will learn everything we need to know about test scenarios and scenario testing. Before starting with test scenarios and scenario testing, let's first understand – what is a scenario?

A scenario is a credible and coherent story about how someone can use an application.

What is a Test Scenario?

A Test Scenario is a statement describing the functionality of the application to be tested. It is used for end-to-end testing of a feature and is generally derived from the use cases.

Test scenarios can serve as the basis for lower-level test case creation. A single test scenario can cover one or more test cases. Therefore a test scenario has a one-to-many relationship with the test cases.



As an example, consider a test scenario – "Verify that the user is not able to login with incorrect credentials". Now, this test scenario can be further broken down into multiple test cases like-

1. Checking that a user with the correct username and incorrect password should not be allowed to log in.
2. Checking that a user with an incorrect username and correct password should not be allowed to log in.
3. Verifying that users with incorrect usernames and incorrect passwords should not be allowed to log in.

Obviously, the test cases will have a well-defined format, explained in this post – Test case template.

Also, check – Difference b/w a test case and test scenario.

What is Scenario Testing?

Scenario testing is a type of testing carried out using scenarios derived from the use cases. Also, using scenario testing, complex application logic can be tested using easy-to-evaluate test scenarios.

Advantages of Test Scenarios

- Scenario testing can be carried out relatively faster than testing using test cases.
- It can ensure good test coverage since the test scenarios are derived from user stories.
- It saves a lot of time. Hence, these are better with projects having time constraints.

Test Scenario Template

A Test Scenario document can have the below fields-

- **Module** – The module or the component of the application.
- **RequirementId** – This field is optional and can be linked to the SRS.
- **TestScenarioId** – This field is the identifier of the test scenarios.
- **Description** – The description field describes the purpose of the test scenario.

Best Practices for Writing Test Scenarios

- Should be easy to understand.
- Easily executable.
- Should be accurate.
- Traceable or mapped with the requirements.
- Should not have any ambiguity.

Test Scenario Examples

Below is the list of test scenarios that are frequently asked in software testing interviews. In these test scenario examples, we are covering scenarios related to UI, functionality, non-functional requirements as well as negative test scenarios.

Although there can be numerous scenarios for any given application, we have limited the scenarios to the most basic and generic functionalities.

- Test Scenarios of ATM Machine
- Test Scenarios of Bike
- Test Scenarios of Calculator
- Test Scenarios of Car
- Test Scenarios of Chair
- Test Scenarios of Coffee Vending Machine
- Test Scenarios of DateField
- Test Scenarios of Door
- Test Scenarios of E-commerce Website
- Test Scenarios of Facebook

- Test Scenarios of Fan
- Test Scenarios of Flight Reservation System
- Test Scenarios of Gmail
- Test Scenarios of Google Search
- Test Scenarios of Hospital Management System
- Test Scenarios of Keyboard
- Test Scenarios of Kindle
- Test Scenarios of Lift
- Test Scenarios of Login Page
- Test Scenarios of Mobile Phone
- Test Scenarios of Mouse
- Test Scenarios of Microwave Oven
- Test Scenarios of Notepad
- Test Scenarios of Online Examination System
- Test Scenarios of Pen
- Test Scenarios of Pencil
- Test Scenarios of Registration Page
- Test Scenarios of Remote Control
- Test Scenarios of Stapler
- Test Scenarios of Table
- Test Scenarios of Triangle
- Test Scenarios of TV
- Test Scenarios of Wrist Watch
- Test Scenarios of Water Bottle
- Test Scenarios of Whatsapp
- Test Scenarios of White Board
- Test Scenarios of White Board Marker
- Test Scenarios of Youtube

What is a Test Case? How to write test cases?

Software testing of an application includes validating its functional as well as non-functional requirements. For validating these requirements, software testers are required to create effective test cases using different white box and black box test design techniques.

What is a Test Case?

A test case is a set of conditions for evaluating a particular feature of a software product to determine its compliance with the business requirements.
A test case has pre-requisites, input values, and expected results in a documented form that cover the different test scenarios.

Once the test cases are created from the requirements, it is the job of the testers to execute those test cases. The testers read all the details in the test case, perform the test steps, and then based on the expected and actual result, mark the test case as **Pass** or **Fail**.

Test Case Attributes

Let's check the different attributes of a test case that comprise a test case and make them more reliable, clear, and concise avoiding or reducing any sort of redundancy.

1. **TestCaseId** – A unique identifier of the test case.
   It is a mandatory field that uniquely identifies a test case e.g. TC_01.

2. **Test Summary** – One-liner summary of the test case.
   This is an optional field. Normally the test cases either have the 'Test Summary' field or the 'Description' field.

3. **Description** – Detailed description of the test case.
   This field defines the purpose of the test case e.g. verify that the user can login with a valid username and valid password.

4. **Prerequisite or pre-condition** – A set of prerequisites that must be followed before executing the test steps.
   For example – while testing the functionality of the application after login, we can have the pre-requisite field as "User should be logged in to the application".

5. **Test Steps** – Detailed steps for performing the test case.
   This is the most important field of a test case. The tester should aim to have clear and unambiguous steps in the test steps field so that some other person can follow the test steps during test execution.

6. **Test Data** – The value of the test data used in the test case.
   For example – while testing the login functionality, the test data field can have the actual value of the username and password to be used during test execution.

7. **Expected result** – The expected result in order to pass the test.
   Based on the test steps followed and the test data used, we come up with the expected result e.g. the user should successfully login and navigated to home page.

8. **Actual result** – The actual result after executing the test steps.
   This field is filled during test execution only. In this field, we write the actual result observed during the test case execution.

9. **Test Result** – Pass/Fail status of the test execution.
   Based on the expected result and the actual result, the test case is marked as passed or Failed.
   Apart from Pass/Fail, we can have other values also like-
   Deferred, when the test case is marked to be executed later, for

some reason.
Blocked, when the test case execution is blocked due to some other issue in the application).

10. **Automation Status** – Identifier of automation – whether the application is automated or not.
    This is an optional field used only when we have automation in the project.

11. **Date** – The test execution date.
    This field helps in keeping track of the different iteration during multiple test execution cycles.

12. **Executed by** – Name of the person executing the test case.
    This field helps when there are multiple team members working on the test execution activity.

For a detailed Test case template in downloadable Xls format, you can also check our tutorial – Test Case Template (Xls).

Test Case Examples

Login Page	Registration Page	E-commerce App	Google Search
GMail	Youtube	Facebook	ATM Machine
Chair	Coffee Machine	DateField	Door
Calculator	Fan	Flight Reservation	Car
Hospital Management	Keyboard	Kindle	Lift
Mobile Phone	Mouse	Microwave Oven	Notepad
Online Examination	Pen	Pencil	Remote Control
Stapler	Table	Triangle	TV
Wrist Watch	Water Bottle	Whatsapp	White Board
Bike	Marker	Forgot Password	Air Conditioner

Test case vs Test Scenario.

A test case is a set of conditions for evaluating a particular feature of a software product. Basically test cases help in determining the compliance of an application with its business requirements.

Whereas, a test scenario is generally a one-line statement describing a feature of the application to be tested. It is used for end-to-end testing of a feature. Usually, it is derived from the use cases.

Let's now see the difference between test case and test scenario.

Test Case vs Test Scenario

Test Case	**Test Scenario**
A test case contains clearly defined test steps for testing a feature of an application.	A test scenario contains a high level documentation, describing an end to end functionality to be tested.
Test cases focus on "what to test" and "how to test".	Test scenarios just focus on "what to test".
These have clearly defined step, pre-requisites, expected results etc. Hence, there is no ambiguity.	Test scenarios are generally one-liner. Hence, there is always possibility of ambiguity during testing.
Test cases can be derived from test scenarios. They have many to one relationship with the test scenarios.	These are derived from use cases.
Test cases are efficient in exhaustive testing of application.	Test scenarios are beneficial in quick testing of end to end functionality of the application.
More resources are required for documentation and execution of test cases.	Relatively less time and resources are required for creating and testing using scenarios.

Well-written test case

To test the application's functionality, a well-described test case is also essential. How to describe a test case correctly?

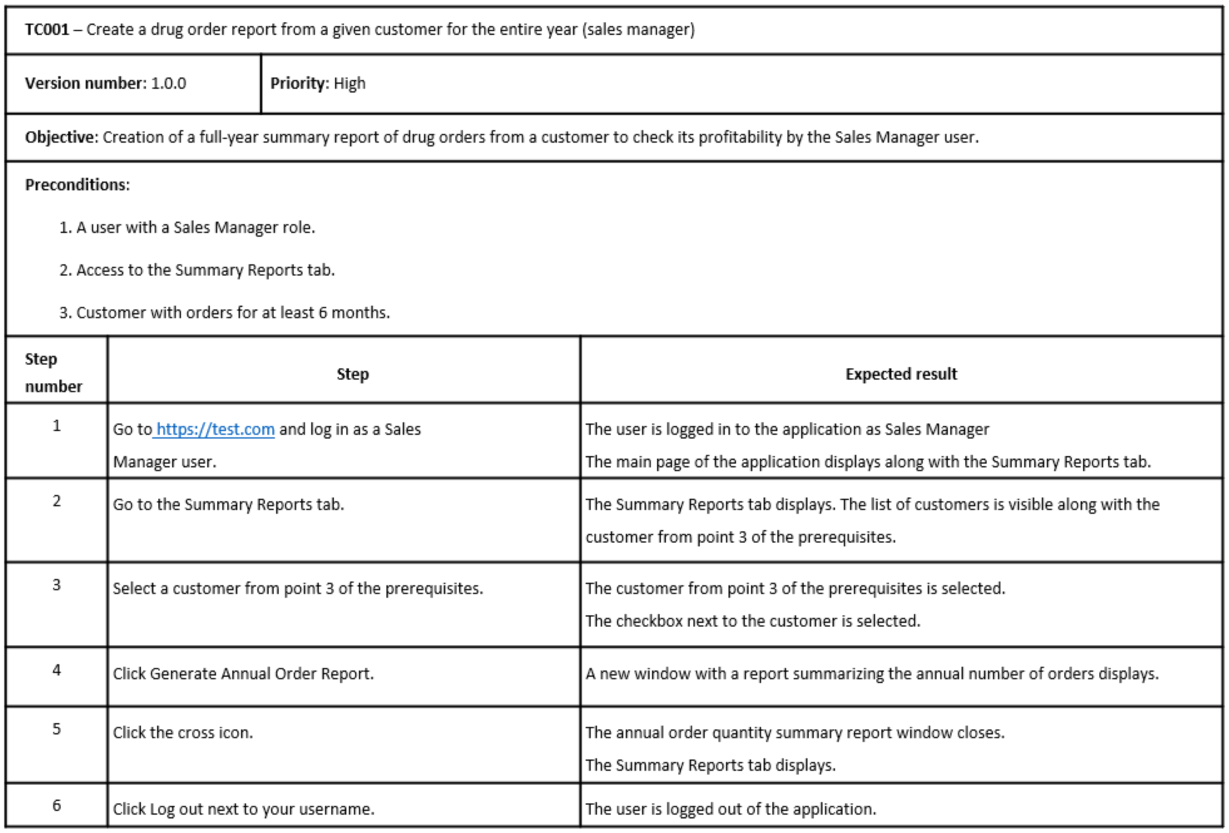
- **ID**. Each test case should have its unique identifier. If you use a test management tool in your project (such as Jira, Testlink), the numbers are generated automatically. If there is no such tool or it is unavailable due to failure, for example, you can use Excel and number the test cases sequentially, TC01, TC02, and so on.
- **Name**. The name should be short, without any details: Order form closing (admin user), for example. The too-long name makes your work unnecessarily difficult. I remember situations early in my testing career when I wanted to name test cases in great detail. One of the examples: "Closing the order form by clicking the cross icon by a user with administrator privileges and redirecting to the main page." Instead of making it easier for myself, I was just wasting time because the name was hard to remember, and it often did not display completely in the test cases list because of too many characters.
- **Objective**. The test objective should be more specific than the name. It should be understood by a person who is not the test case author.
- **Preconditions**. Also called prerequisites. In other words, conditions must be met to proceed with a test case. What roles will be assigned to users, what fields should be completed, what tab should be opened, and similar?

When the Project Manager asks for information regarding the time it takes to complete a given test case, one must always take into account how long it may take to meet the prerequisites, such as filling in the fields and creating appropriate objects. Once I focused too much on steps. There were only a few of them, but I found that it took a long time to meet the prerequisites. So I gave the time too far from what I actually needed to complete the given test cases.

- **Steps.** Steps that should be performed so that the test result can appear in the application. The steps should be described from the end user's point of view. The tester gets into the end user's role as if they were using the product area.
- **Expected result.** The result should appear after executing the given step.
- **Priority.** Defines the test case importance. Allows the Project Manager and manual tester to assess which cases to execute first when time is short and a change needs to be delivered quickly.
- **Version number.** We have consider changes that may be introduced to the given functionality at some point. Then, test cases need to be updated. We then give the new version of the test case a higher number than the previous one.

Test case versioning is more and more often available in tools such as the popular Jira, thanks to which we can at any time review previous versions and recall what changes have been made.

Sample test case



TC001 – Create a drug order report from a given customer for the entire year (sales manager)		
Version number: 1.0.0	**Priority:** High	
Objective: Creation of a full-year summary report of drug orders from a customer to check its profitability by the Sales Manager user.		
Preconditions: 1. A user with a Sales Manager role. 2. Access to the Summary Reports tab. 3. Customer with orders for at least 6 months.		
Step number	**Step**	**Expected result**
1	Go to https://test.com and log in as a Sales Manager user.	The user is logged in to the application as Sales Manager The main page of the application displays along with the Summary Reports tab.
2	Go to the Summary Reports tab.	The Summary Reports tab displays. The list of customers is visible along with the customer from point 3 of the prerequisites.
3	Select a customer from point 3 of the prerequisites.	The customer from point 3 of the prerequisites is selected. The checkbox next to the customer is selected.
4	Click Generate Annual Order Report.	A new window with a report summarizing the annual number of orders displays.
5	Click the cross icon.	The annual order quantity summary report window closes. The Summary Reports tab displays.
6	Click Log out next to your username.	The user is logged out of the application.

The test case should be checked by another tester who can point out correct suggestions, catch typos, and similar.

The test case quality should be at the highest level, especially in the so-called validated projects. They are additionally checked by a validator – a person responsible for their degree of accuracy. In such projects, the form of writing test cases is usually imposed by the validator, and, from my experience, you have to adapt to the form desired in the project.

Test cases coverage

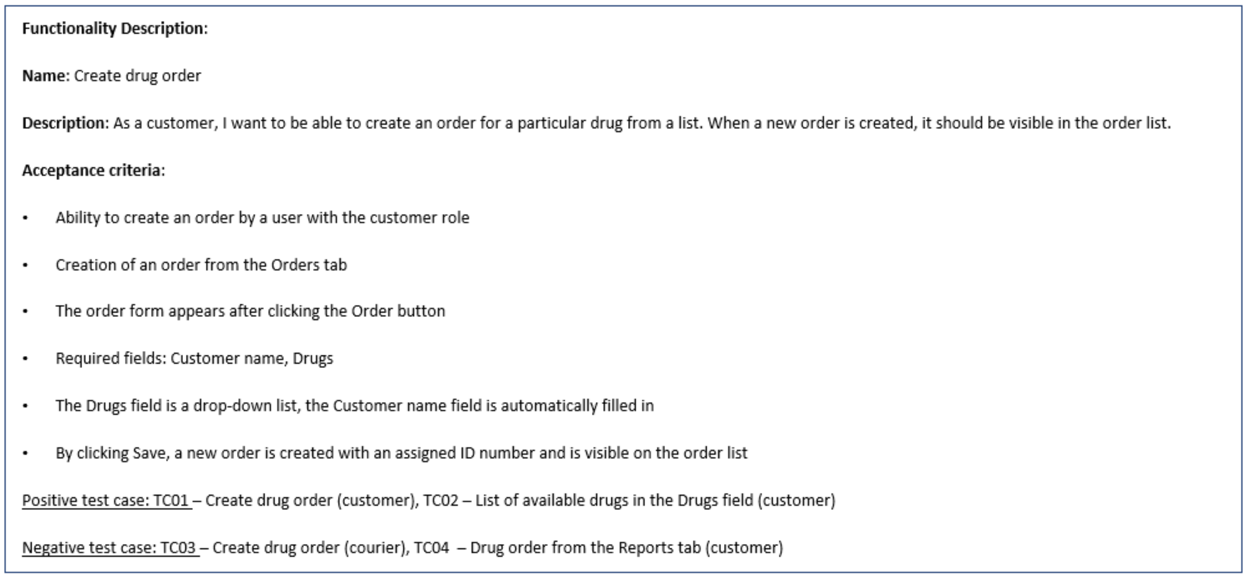
In functional testing, we should include positive as well as negative test cases.

- Positive – show that the given change works as described and intended in the application.

- Negative – are used to check what happens as a result of undesired functionality operation, after entering bad data, for example.

Let me describe this with an example.

Types of test cases



Functionality Description:

Name: Create drug order

Description: As a customer, I want to be able to create an order for a particular drug from a list. When a new order is created, it should be visible in the order list.

Acceptance criteria:

- Ability to create an order by a user with the customer role
- Creation of an order from the Orders tab
- The order form appears after clicking the Order button
- Required fields: Customer name, Drugs
- The Drugs field is a drop-down list, the Customer name field is automatically filled in
- By clicking Save, a new order is created with an assigned ID number and is visible on the order list

<u>Positive test case: TC01</u> – Create drug order (customer), TC02 – List of available drugs in the Drugs field (customer)

<u>Negative test case: TC03</u> – Create drug order (courier), TC04 – Drug order from the Reports tab (customer)

Test case vs. test scenario

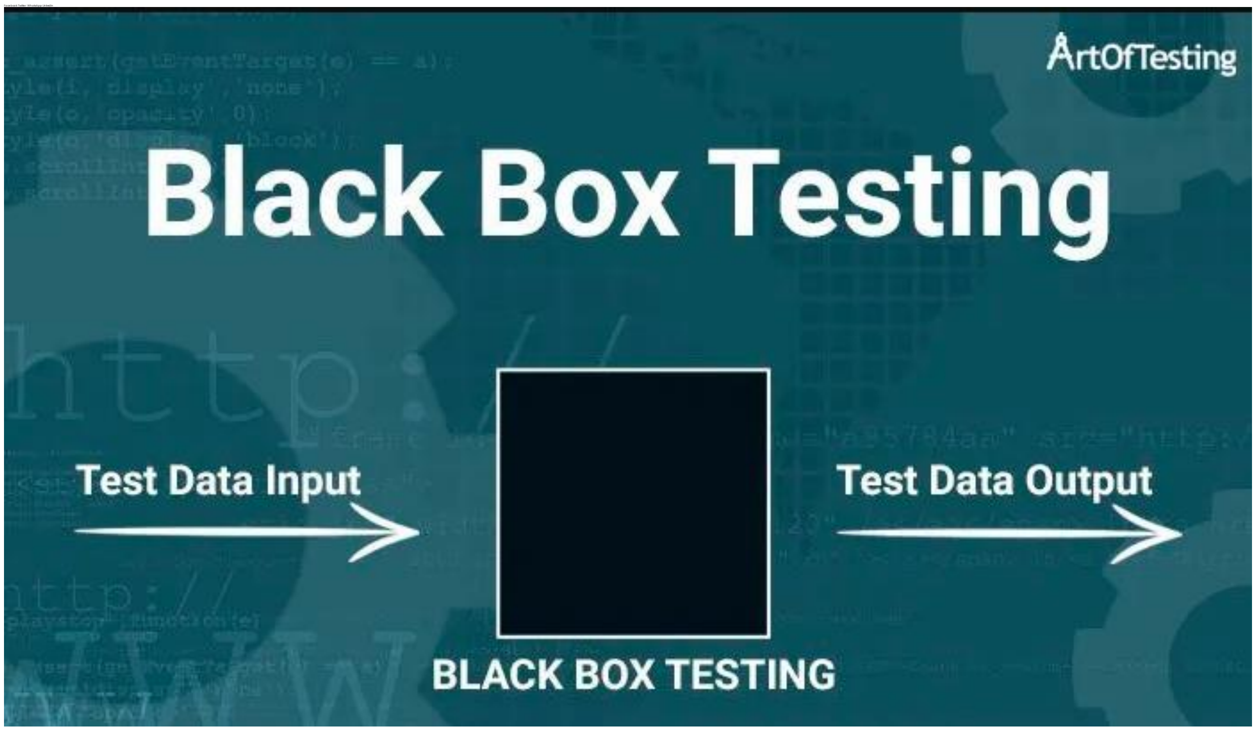
You may find the terms test case and test scenario alternately used, which may be confusing. The easiest way to distinguish them is to compare the scale/scope: a test case covers a part of the application process, while a test scenario usually covers the entire process, meaning several test cases, both negative and positive.

Differences between test case and test scenario

Test scenario = { test case1, test case2, test case3...}

Drug orders = {TC01 – Create drug order (customer); TC02 – List of available drugs in Drugs field, TC03 – Create drug order (courier); TC04 – Drugs order from Reports tab (customer), TC05 – Edit drug order (customer), TC06 – Delete drug order (customer)}

Black Box Testing



Black box testing is also referred to as **specification-based testing**. It involves performing testing based on the specification of the system under test. Unlike white-box testing, the knowledge of the internal architecture and the application code is not required in black-box testing.

Black Box Testing Definition

Black box testing is the type of testing in which an application is tested based on its requirements specifications without the need for knowledge of its internal architecture.

Features of Black Box Testing

- It tests both functional as well as non-functional requirements of the application.
- Knowledge/access to the coding/design/internal architecture of the software is not required.

- Testers can work independently from developers thus ensuring unbiased and end-user centric testing.

httpv://www.youtube.com/watch?v=Sqkda4c\u002d\u002dEI

How to do black-box testing?

After learning black box testing techniques, let us learn the different steps involved in a typical black-box test.

- The first step is to check the requirement specifications provided by the application. The requirements should be provided in a properly documented SRS file.

- Tester collects the different positive test scenarios and negative test scenarios to verify if the system under test processes them correctly. This ensures good test coverage.

- The test cases are executed and the output is validated against the expected results. This process is to mark the pass or fail of the test result.

- The failed test cases are sent back to the development team to fix the bugs.

- After the fix, a retest is conducted to check and ensure all the test cases run successfully.

Advantages of black-box testing

1. Tests are performed from the user's point of view. So, there is higher chance of meeting customer's expectations.

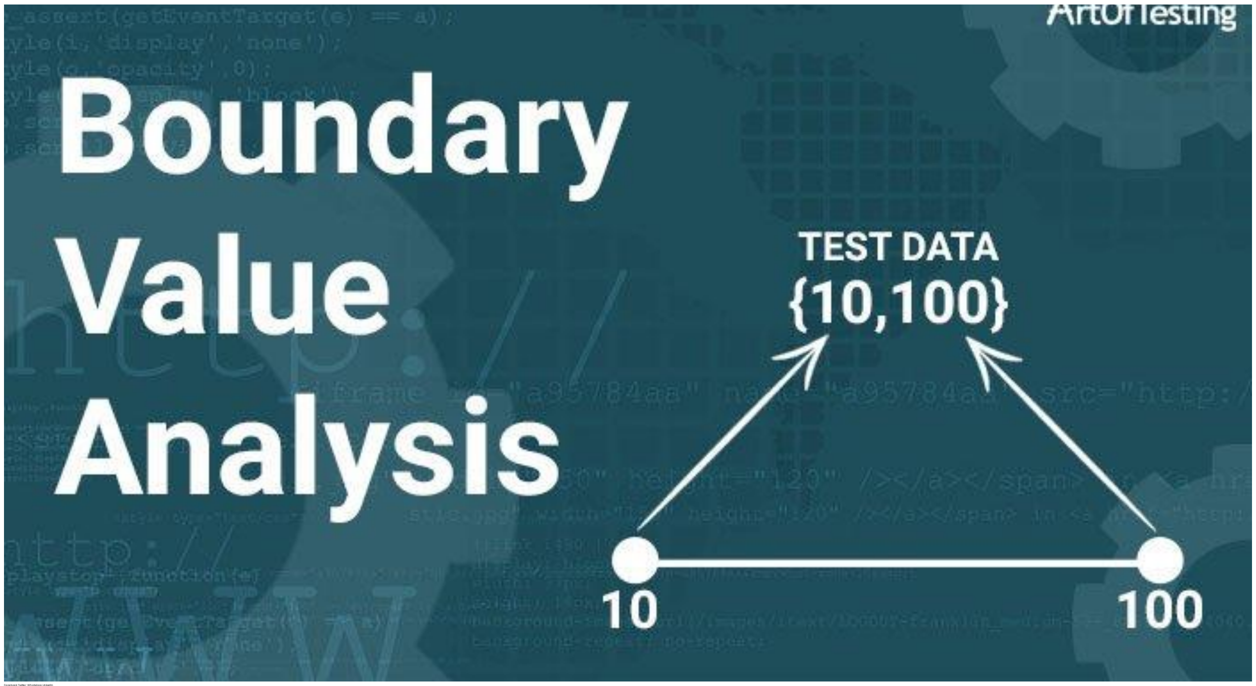
2. There is unbiased testing as both the tester and the developer work independently.

3. It is suitable for the testing of very large systems.

4. There is no need for any technical knowledge or language specification.

5. Test cases can be designed as soon as the requirements are finalized.

Boundary Value Analysis



Software testing or rather exhaustive software testing is a very time and resource-intensive activity. In order to effectively test any application in the best possible time and with optimal resources, we use different test design techniques. One such technique is boundary value analysis.

In this article, we will explore this testing technique along with an example and also check its advantages and disadvantages.

What is boundary value analysis?

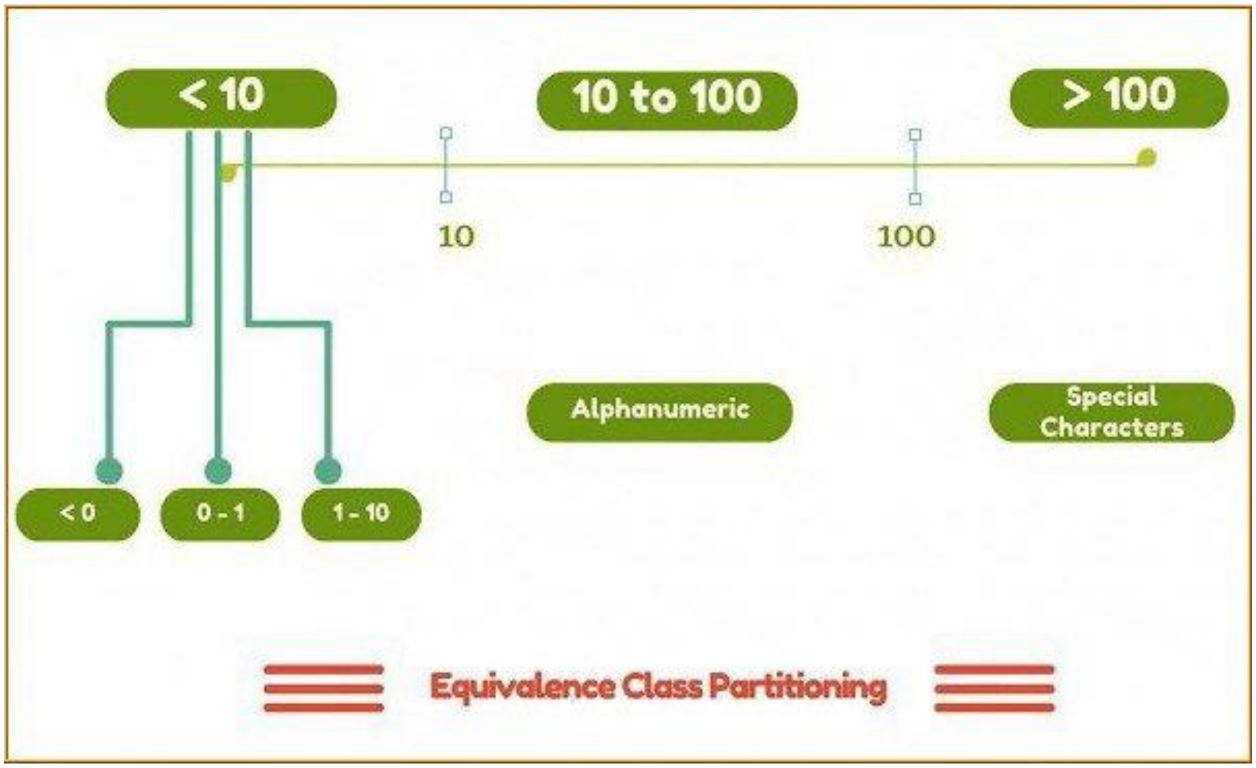
Boundary value analysis is a black-box testing technique. It is closely associated with equivalence class partitioning. In this technique, we analyze the behavior of the application with test data residing at the boundary values of the equivalence classes.

By using the test data residing at the boundaries, there is a higher chance of finding errors in the software application.

Boundary Value Analysis Example

Let's consider the same example we used in the equivalence partitioning tutorial. An application that accepts a numeric number as input with a value between 10 to 100.

While testing such application, we will not only test it with values from 10 to 100 but also with other sets of values like – less than 10, greater 10, special characters, alphanumeric, etc.



For increasing the probability of finding errors instead of picking random values from those classes, we can pick the values at the boundaries like below-

Equivalence Classes	**Test Data using Boundary Value Analysis**
Numbers between 10 to100	10, 100
Numbers less than 10	9
Numbers greater than 100	101

Advantages of Boundary Value Analysis

1. It is easier and faster to find defects using this technique. This is because the density of defects at boundaries is more.

2. Instead of testing will all set of test data, we only pick the one at the boundaries. So, the overall test execution time reduces.

Disadvantages of boundary value analysis

1. The success of the testing using this technique depends on the equivalence classes identified, which further depends on the expertise of the tester and his knowledge of the application. Hence, incorrect identification of equivalence classes leads to incorrect boundary value testing.

2. Applications with open boundaries or applications not having one-dimensional boundaries are not suitable for this technique. In those cases, other black-box techniques like “Domain Analysis” are used.

White Box Testing

Testing is essential to software development to ensure the delivery of high-quality, error-free software products. Testers employ different software testing techniques to identify different issues, defects, and errors in software products. Among all white box testing is one. It is concerned with evaluating a software's internal structure and implementation details.

This blog post will discuss the essential aspects of white box testing in software engineering.

What is White-box testing?

White box testing is a software testing technique that tests a system's internal design, source code structure, data structures used, and working details. Its primary objective is to improve the software's design, input-output flow, usability, and security. It is also called transparent testing, structural testing, and glass box testing.

Implementing this testing technique requires testers to know the system's code, architecture, and implementation details. Using this knowledge, they create test cases and execute them to verify the system's correctness at the code level. Hence, it is also known as code-based testing.

Generally, **developers** perform white box testing. They have complete knowledge of the software's source code and internal workings. However, in some cases, **quality assurance (QA) professionals and testers** who understand complex code can also do it.

This testing technique is called 'White Box' because developers or testers peek into a system's internal workings from its outer shell.

It applies to the first two levels of software testing – unit testing and integration testing. Unit testing validates each software module independently. Subsequently, integration testing combines unit-tested modules logically and tests their interaction or communication.

Features

- **Access to the Source Code:** White box testing provides access to the software’s source code. This helps validate individual functions and modules.
- **Code Coverage Analysis:** Code coverage is a metric that determines the amount of code executed during testing. White box testing analyzes code coverage and uncovers untested source code areas.
- **Detecting Logical Errors:** It helps identify logical errors like infinite loops and incorrect conditional statements.
- **Code Optimization:** It detects performance issues, areas of code that need to be improved, and other issues. Developers or testers work to fix these issues and optimize the source code.
- **Security Testing:** Developers or testers can access the software’s source code and know its internal workings. Hence, they can identify security vulnerabilities.

What To Verify in White Box Testing?

White box testing in software testing evaluates the software’s source code to verify the following parameters:

- Internal security vulnerabilities.
- Each object, function, and statement of the source code individually.
- The functionality of conditional loops.
- Broken, incomplete, and poorly structured code paths.
- The input and output flow.

In short, this testing technique validates a software’s working flow. It involves providing a set of inputs and comparing the expected and actual outputs. If the actual output does not match the expected one, it results in an error or bug.

White Box Testing Example

Now, we know that white box testing aims to verify the code structure, such as loop statements, conditional statements, decision branches, etc. We'll understand it with a simple example. Consider the following code:

```
Test (a, b)
{
 int n;
 if (n % 2 == 0)
  {
   print("Even number")
  }
 else
  {
   print("Odd Number")
  }
}
```

To validate this code, we have the following two test cases:

- n = 25
- n = 50

For the first test case, n = 25, the 'if' condition does not hold true. Hence, the program flow moves to the 'else' block and prints the statement inside it. For the second test case, n = 50, the 'if' condition holds true, and the statement inside it gets executed.

This way, white box testing has exercised each line of an application's source code and uncovered potential code-level errors.

White Box Testing Techniques

The different types of white box testing techniques are as follows:

1. Statement Coverage

This technique requires traversing and testing each statement in the source code at least once. As a result, the entire source code gets exercised.

The statement coverage determines the percentage of the source code a specific set of test cases exercises. The formula for statement coverage is:

Statement Coverage = (Number of Statements Executed / Total Number of Statements) * 100

2. Decision Coverage/Branch Coverage

The best example of a branch (decision point) in programming is the 'if' statement. It has two branches – True and False. The branch coverage technique ensures that each branch in the source is executed at least once.

Branch coverage implies the percentage of branches or decision points executed during testing.

Branch Coverage = (Number of executed branches / Total number of branches) * 100%

3. Condition Coverage

Condition testing involves testing the individual conditions for both TRUE and FALSE outcomes. So, getting 100% condition coverage requires exercising each condition for both TRUE and FALSE results. For n conditions, we will have 2n test scripts.

The primary aim of condition coverage is to determine the output of each condition in the source code. However, it tests only those conditions with logical operands whose outcome is either true or false.

4. Multiple Condition Testing

It aims to test all the possible combinations of every condition in a branch. Let us understand this with an example.

Consider the following code:

```
if (A||B)
   print C
```

The test cases for the above code will be:

1. A=TRUE, B=TRUE
2. A=TRUE, B=FALSE
3. A=FALSE, B=TRUE

4. A=FALSE, B=FALSE

Our example has 2 conditions – A and B, and 4 test cases. If there were 3 expressions, the number of test cases would be 8.

Hence for 100% coverage, we will have 2^n test scripts. This is very exhaustive, and it is very difficult to achieve 100% coverage.

5. Path Testing

Path testing ensures that all possible paths in the source code are executed at least once. It involves creating a control flow graph using the source code or flowchart. Later, it determines the Cyclomatic complexity, which refers to independent paths. So, testers create minimal test cases for such independent paths.

Path Coverage = (Number paths exercised / Total Number of paths in the program) x 100 %

6. Loop Testing

Loops are common programming constructs and are used in most large programs. Testing loops is essential, as there are high chances of errors occurring at the start or end of loops. Hence, performing loop testing uncovers bugs or errors in any specific loop. The primary error encountered in loops is wrong indexes.

Types of White Box Testing

Here are different types of white box testing:

- **Unit Testing:** This is the first level of software testing. It tests an application's every module, called a unit, individually for its correctness. It ensures that each component functions as expected.

- **Integration Testing**: This comes after integration testing. It combines unit-tested components logically and validates the interaction between them. It aims to uncover any errors in the interaction of components.

- **White Box Penetration Testing:** Testers have complete access to an application’s source code and network, IP, and server data, including passwords and maps. The primary goal of penetration testing is to uncover areas of the source code with security vulnerabilities.

- **White Box Mutation Testing:** As the name suggests, mutation testing depends on alterations. Testers perform minute modifications to the source code to check whether the execution of test cases on it uncovers any bugs. If test cases pass, it indicates an error in the source code. However, if test cases fail, the source code is error-free.

Advantages and Disadvantages of White Box Testing

Let us now shed light on the advantages and disadvantages of white box testing.

Advantages

- White box testing is comprehensive and detailed as it exercises every line of the source code.
- It identifies potential hidden errors, defects, and security vulnerabilities. Fixing them requires removing some lines of source code, which results in code optimization.
- It ensures that the source code complies with the coding standards and is performance-optimized.
- Even if the GUI is unavailable, testing starts early in the software development life cycle (SDLC).
- Test cases are easy to automate.
- The source code transparency helps determine the exact type of input data required for testing.
- Testers or developers can create test cases that can ensure maximum test coverage.

Disadvantages

- White box testing requires in-depth programming knowledge to understand and analyze a system's source code and create test cases around it.
- It primarily focuses on testing the system's internal workings and misses out on external issues.
- Large applications require a lot of time to undergo white box testing due to their lengthy source codes.
- A small change in the source code requires writing test cases again.
- There are strong chances of resulting in production errors.

White Box Testing Tools

Here is a list of some commonly used white box testing tools:

1. **Veracode:** It provides a suite of tools that help identify and fix flaws in applications developed using different programming languages, such as .NET, C++, Java, etc. You can also test desktop and mobile applications for security.

2. **EclEmma**: It is a free code coverage tool for Java applications. It was designed to run tests and analyze results within the Eclipse workbench.

3. **JSUnit.net:** JSUnit is a component of JUnit, a unit testing framework for Java applications. JSUnit is an open-source unit testing tool for JavaScript testing. It is available under GNU General License 2.0.

4. **NUnit:** It is a testing framework developed in C# to perform data-driven testing on .NET applications. It supports the parallel execution of tests without any manual intervention.

5. **CppUnit:** It is also the component of JUnit as JSUnit. It is available for unit testing C++ applications.

Black box vs White box testing

The difference between black-box testing and white-box testing is one of the most common testing interview questions. Both are equally important and performed according to the situation. Here are a few differences to bring clarity to both techniques.

Black box testing	**White-box testing**
Any knowledge of implementation is also not required.	The whole internal working of the SUT is verified by a tester that has complete knowledge of it.
Aims at validating the functional requirements of the software.	Aims at code optimization.
This testing is usually done by the software testers.	This testing is usually done by software developers as they have knowledge of internal architecture and implementation of the application.
It is less time-consuming.	It is comparatively more time-consuming.
Types – Functional Testing, Non-functional testing, Regression Testing, etc.	Types of white box testing – path testing, loop testing, condition testing, etc.

What is Automation Testing?

Facebook Twitter WhatsApp LinkedIn

What is Automation Testing?

Automation testing is a type of software testing that involves automated test case execution using an automation tool.

So, basically, it automates the manual testing process. The tester writes test scripts and then runs the test scripts either on-demand or schedule them for periodic executions. This reduces the overall testing time, thus helping in faster product releases.

What to Automate?

Now that we know what exactly is Automation testing, let's check **which test cases to automate**. Or, what all test cases are ideal candidates for automation.

1. Test cases that test critical functionality of the application

For example for an e-commerce application, the critical functionality would be the product discovery via search and category pages, add to cart and then buy functionality. So, these test cases should be first chosen. Test cases for add to wishlist and notify me etc should be of lower priority. Hence should be picked accordingly for automation.

2. Test cases that require repeated test execution with a large dataset

There are many test cases or application flow that require performing an action again and again. Such test cases are also ideal candidates for automation as automation reduces a considerable amount of testing effort.

Take an example of a search feature of an application. If we can automate the flow to search with a search term and then verify that search results. Then we can run the same script again and again with different types of search terms like – single word, multi-word, alphanumeric, with special characters, with foreign language characters, etc.

3. Tests that are time-consuming

Workflows that require a considerable amount of time to execute and set up are also ideal candidates for automation.

Let's take the example of e-commerce application forward. If some test cases require the set up of multiple products and then performing some operation on those products. Such test cases when automated not only reduce the test execution time but also, free the manual testers of the redundant task. In addition. helping them focus on other exploratory testing activities.

4. Test cases that require parallel execution

There are scenarios that require checking the concurrent access to the application e.g. in the case of performance testing with multiple users. In such cases, either manual testing is not feasible or it would require a lot more resources to test the particular scenarios. In those cases, automated scripts help by creating concurrent requests and collating results in one place.

What not to Automate?

It is also important to understand, what sort of test cases cannot or rather should not be automated.

1. UI test cases

GUI or the Graphical User Interface test cases should best be left for manual testing or human validation. This is because even with the slightest change in the UI, the test cases would fail. Moreover, it is also very difficult to create reliable UI test cases across multiple devices and screen resolutions.

2. Usability test cases

Rather than 'should not' it is the case of 'cannot' automate. Usability test cases, test application's ease of use by different sets of users which, with the current technology is not possible to automate.

3. Functionalities that are rarely used and take time for scripting

It is good to automate complex scenarios but investing your effort in scenarios that would rarely be used doesn't provide a good Return on Investment.

4. Exploratory testing

Exploratory testing requires learning about the application on the fly and simultaneous testing. Hence, it is not possible to automate exploratory testing scenarios.

When should we Automate?

After defining, all the capabilities of the **automation suite** during Test Planning, we can begin the **automation framework** creation activity in parallel with the development team. But the scripting of the test cases should be done at the right time.

For better automation ROI and to avoid any rework – **scripting of test cases should start when the application is stable and frequent changes in the application are not anticipated.**

Why Automated Testing?

Following are some of the advantages or benefits of automation testing-

- Automation testing **reduces the overall test execution time**. Since automated test execution is faster than manual test execution.
- It **reduces the cost and resource requirements** of the project. This is because the script created once can be made to

run any number of times as long as there is no change in the application.

- Helps in **working with a large set of input** which isn't be feasible with manual testing.
- Helps in creating a **Continuous Integration** environment wherein, after each code push, automatic test suite execution takes place with the new build. Using CICD tools like Jenkins, we can create Jobs that execute the test cases after the deployment of a build and mails the test results to the stakeholders.

When not to Automate?

Let's see some scenarios, where it is not advisable to do automated testing along with some disadvantages of automation.

- **Lack of expertise of the automation tool** – Lacking expertise in the automation tool and\or programming language to create robust scripts is one of the primary reason which can lead to not using the tool to its full potential. Factors like these lead to the failure of automation testing.
- **Incorrectly chosen test cases** – The success of automation testing heavily depends on the test cases chosen for automation. Incorrectly chosen tests lead to wastage of resources and time invested in automation.
- **Applications with frequent changes** – Choosing test automation for an application with frequent changes requires constant maintenance of the test scripts, which at times might not give the desired ROI.
- **Inefficiently written test scripts** – Test scripts with limited or inadequate validations can lead to false-positive test results. These false-positive results conceal the underlying defects which

could have been easily captured if validated manually or scripted in a better way.

Popular Test Automation Testing Tools

The software market is full of paid and free test automation tools. Based on the various factors like – project requirement, budget, the expertise of the resources, etc, we should choose the right tool, suited for our needs. The following are some of the most popular test automation tools in the market.

Selenium

Selenium is an open-source test automation tool. It has a very large and active community. It has the maximum market share among all the popular tools and supports scripting in multiple languages – Java, Python, Ruby, Javascript, C#, etc. Download link – Selenium Download

Katalon Studio

A fairly new tool but rapidly getting popular due to record and playback features along with scripting for more technical users. It is free but not open source.

UFT One

A paid tool by Microfocus that can be used for automation of both Web and Windows applications. It supports scripting in VBScript only.

TestComplete

A paid tool provided by Smartbear can be used for automation of Web, Mobile as well as Desktop applications.

Tosca

It is a paid tool by Tricentis that provides record and playback features for automating web applications, APIs, and windows applications. It is considered one of the most popular codeless automation tools that can completely eliminate the need for scripting.

Watir

Watir (Web Application Testing In Ruby) is an open-source automation tool that can automate web applications in Ruby. It has a watir-webdriver component that is based on Selenium.

Appium

It can be considered as Selenium for mobile applications. Just like Selenium, it is open-source and has a large user base.

TestProject

TestProject is a free and community-powered automation testing tool by Tricentis. It can be used for the automation of both web and mobile applications.

Ranorex

A paid tool with record and playback features. Using this we can automate android, ios and Windows applications. Along with record and playback, it also supports scripting in using C# and VB scripts.

For more details, you can check our post – Top Test Automation Tools containing in-depth review, tool's features, download link, and our recommendations of the different tools.

Tutorials on Automation Testing Tools

Selenium	Katalon	Cucumber	TestNG

What is Regression Testing? Test Cases (Example)

ByThomas HamiltonUpdatedMay 13, 2023

What is Regression Testing?

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression Testing is nothing but a full or partial selection of already executed test cases that are re-executed to ensure existing functionalities work fine.
This testing is done to ensure that new code changes do not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

Why Regression Testing?

There is a **need for regression testing** whenever the code is changed, and you need to determine whether the modified code will affect other parts of the software application. Moreover, regression testing is needed when a new feature is added to the software application. Regression tests may also be performed when a functional or performance defect/issue is fixed.

When can we perform Regression Testing?

Here are the scenarios when you can perform regression testing.

New functionality is added to the application: This happens when new features or modules are created in an app or a website. The regression is performed to see if the feature is working properly.

In case of change requirement: When any significant change occurs in the system, regression testing is used. This test is done to check if these shifts have affected other features.

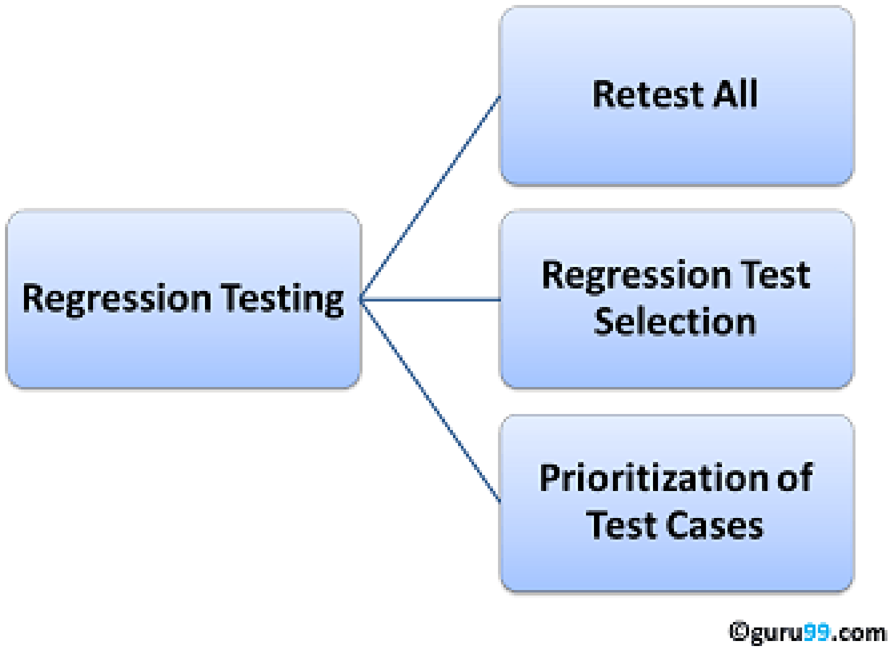
After a defect is fixed: The developers perform regression after fixing a bug issue in any functionality. This is done to determine if the changes made while fixing the issue have affected other related features.

Once the performance issue is fixed: After fixing any performance issues, regression testing is done to see if it has affected other functionalities.

While integrating with a new external system: Regression testing is required whenever the product integrates with a new external system.

How to do Regression Testing in Software Testing

In order **to do Regression Testing** process, we need to first debug the code to identify the bugs. Once the bugs are identified, required changes are made to fix it, then the regression testing is done by selecting relevant test cases from the test suite that covers both modified and affected parts of the code.
Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work incorrectly. Therefore, Regression Testing becomes necessary. Regression testing, and specifically various regression testing techniques, can be carried out for effective software quality assurance:



Retest All

This is one of the methods for Regression Testing, specifically employing a regression testing suite, in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

Regression Test Selection

Regression Test Selection is a technique in which some selected test cases from test suite are executed to test whether the modified code affects the software application or not. Test cases are categorized into two parts, reusable test cases which can be used in further regression cycles and obsolete test cases which can not be used in succeeding cycles.

Prioritization of Test Cases

Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

Regression Testing Tools

If your software undergoes frequent changes, regression testing costs will escalate. In such cases, Manual execution of test cases increases test execution time as well as costs. Automation of regression test cases is the smart choice in such cases. The extent of automation depends on the number of test cases that remain re-usable for successive regression cycles.

Following are the most important tools used for both functional and regression testing in software engineering:

1) **testRigor**
2) **Avo Assure**
3) **Avo Assure**
4) **Selenium**
5) **Quick Test Professional (QTP)**