Unit 1 – Manual Testing:

1. Introduction to Software Testing:

What is Software Testing?

Software testing is a process of executing the application with the intent of finding the defects by comparing the output behavior of the application with the expected behavior (requirement).

In other words, it’s comparing the actual behavior of an application with the expected behavior.

Why Software Testing?

Humans make mistakes all the time!!

Software testing is really required to point out the defects and errors that were made during the development phases

We, humans, cannot identify our mistakes in work done by us. We should get someone else to check our work because another person may identify the mistakes done by us. In the same way, software developers may not identify the mismatches in a program or application implemented by them which can be identified by another department called Software Test Engineer.

Benefits of Software Testing

“Software testing helps in finalizing the software application against business requirements.”

Software testing makes sure that the testing is being done properly and hence the system is ready for the customers to use.

Below are a few benefits of software testing.

- Finding the defects before delivery
- Gain confidence in the quality
- To Prevent defects
- Ensure the requirements are delivered to the client.

What is defect

A defect is a deviation or mismatch from the requirements.

When the actual result deviates from the expected result while testing a software application or product then it results in a defect. Hence, any deviation from the specification mentioned in the functional specification document is a defect. In different organizations, it’s called differently like bugs, issues, incidents, or problems.

Project vs Product:

The project is developed for a single customer on his own requirements by the software companies and the project will be used by the customer only.

The product is developed for multiple customers on their consolidated requirements by the software companies and the product will be used by all customers.

What is Software Development Life Cycle

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace, and alter or enhance specific software”

Why Software Development Life Cycle

SDLC ensures success in the process of software development.

Phases of Software Development Life Cycle

- Initial
- Analysis
- Design
- Coding
- Testing

Testing Methods:

We have two types of testing methods, they are:

1. Static Testing

Static Testing is also known as Verification. It is a method of verifying files and documents. It ensures that we are developing the correct product. It also verifies the requirements we have. In this method, test engineers will carry out activities like Reviews, Inspections, Walkthroughs, etc.

2. Dynamic Testing

Dynamic Testing is also called Validation. It is a dynamic process of testing the software product. It validates whether the developed product is right or not.

How to do Manual Testing?

Follow these steps to do manual testing:

Step 1: First, review all the documents related to the software, for selecting the testing areas.

Step 2: Analyse all the required documents to get all the requirements mentioned by the end user.

Step 3: Build test cases as per the requirement document.

Step 4: Execute all the test cases manually by using white-box testing and black-box testing.

Step 5: If bugs are detected, report them to the development team.

Step 6: When the developer fixes the bug, retest it.

Advantages and Disadvantages: Manual Testing

Advantages of Manual Testing:

1. Manual testing detects almost every bug and issue of the software application.
2. Through manual testing, software testers can access visual components like layout, text, and other components. They can also identify the UX and UI issues.
3. It is suitable if we are making some unplanned changes to the applications as they can be adopted easily.
4. As we do not use any high-level skills or type of tools, it will have a low cost of operations.

Disadvantages of Manual Testing:

1. The primary disadvantage of manual testing is it is time-consuming.
2. Through manual testing, it is not easy to discover the colour combination and size difference of the GUI objects.
3. Performance testing and Load testing are impossible in manual testing.
4. In Manual Testing, Regression test cases are time-consuming.

Roles and Responsibilities of a Manual Tester

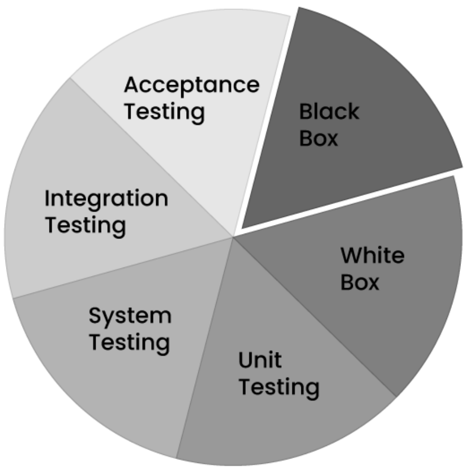
Following are the roles and responsibilities of a Manual Tester:

- Gathering and analyzing client requirements.
- Establishing the test environment for executing the test cases.
- Analysing and running the test cases.
- Arranging Review Meetings.
- Communicating with the users to understand the product requirements.
- Interacting with the Test Manager.
- Defect Tracking.
- Evaluating the testing results on bugs, errors, database impacts, and usability.
- Preparing the reports on all the activities carried out while testing the software and reporting to design time.

SDLC vs. STLC

Basis of Comparison	**SDLC**	**STLC**
Definition	SDLC is mainly connected to software development, which indicates that it is the process of developing a software application.	STLC is primarily connected to software testing, which indicates that it is the process of testing software.
Full-Form	The full form of SDLC is the Software Development Life cycle.	The full form of STLC is the Software Testing Life cycle.
Participants	In the Software development process, many developers will take part.	In the Software testing process, a small number of testers will be involved.
Aim	The aim of SDLC is to finish the development of software.	The aim of STLC is to complete the software testing.
Use	SDLC is useful to develop a good quality software product.	STLC is useful to make the software bug-free.
Designing Phase	According to the requirement analysis, the software development team will build the Low-Level Design (LLD) and High-Level Design (HLD) of the software.	In STLC, the Test Lead or Test Architect will plan the test strategy.
Coding Phase	In the SDLC coding phase, the software developer starts writing the code according to the designed document.	In STLC, the QA team develops the test scenarios for authenticating the product quality.
Testing Phase	In the SDLC testing phase, the software testers will carry out different kinds of testing like integration testing, unit testing, integration testing, etc. The development team will fix the bugs and report them to the tester.	As per the test cases, the tester will perform one round of system and integration testing. While doing the testing, if they detect any bugs, it will be reported and resolved after retesting

Types of Manual Testing:



1. **White box testing:**

It is also known as Clear box, Structural, and Glass box testing. In this type of testing, testers will use the system's internal perspective and programming skills to design the test cases. Generally, this testing is performed at the unit level.

2. **Black box testing:**

In black-box testing, testers assess the functionality of the software application without considering the internal code structure. Black box testing can be applied to all levels of testing like Integration, Unit, Acceptance, and System testing.

3. Acceptance Testing:

Acceptance testing is also known as pre-production testing. Along with the testers, end-users will also perform this testing for validating the functionality of the application. After completing the acceptance testing successfully, formal testing is conducted to decide whether the application is built according to the requirement.

4. Unit Testing:

Unit testing is also known as Component testing or Module Testing. It is carried out to verify whether a particular module or unit of the code is working properly. It is performed by the developers in their environment.

5. System Testing:

System testing is a process of testing the completely integrated application for assessing the system's comfortability with its defined requirements known as System testing or End to End testing. It verifies the entire system to ensure that the application works as planned or not.

6. Integration Testing:

Integration testing is a mechanism of testing the software between two software modules. Integration testing is performed by various approaches called the Top-Down approach, Bottom-Up approach, and Big Bang Approach.

Bug Life Cycle

What is a Bug?

A bug is an error or a flaw of an application that limits the general flow of the application by differing the expected behavior of the application from the actual behavior. The defect takes place when the developer makes a fault while building the application, and when the tester detects this flaw, it is called a defect or bug.

What is the Bug Life Cycle?

The bug life cycle is also known as the defect life cycle, is a bug life cycle that goes through different stages in its life. This life cycle begins once the tester discovers any new bug and ends when the tester marks it as fixed.

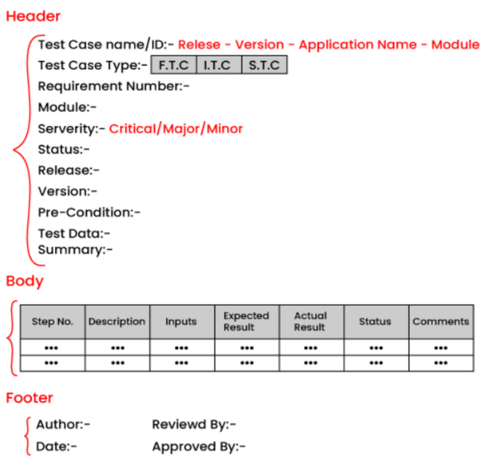
Bug Workflow:

Following are the different states of the Bug life cycle:

1. **New:** This is the first state of the bug life cycle. When the tester finds a new bug, it immediately falls into a "New" state, and in the later stages of the bug life cycle, testing and validation are performed on this bug.
2. **Assigned:** In the Assigned stage, the project lead or project manager of the testing team will assign the newly found bug to the development team to work on the bug.

3. **Open:** In this stage, the developer begins analyzing the bug and fixing it. If the developer finds it inappropriate, it will be moved to four states, namely Deferred, Duplicate, Not a Bug, and Rejected- according to a reason.
4. **Fixed:** After fixing the bug by making the changes, the developer marks the status of the bug as "fixed."
5. **Retest**: In this stage, the tester retests the bug for verifying if the defect is fixed correctly by the developer based on the requirements or not.
6. **Reopen:** If an issue remains in the bug, then the testing team will assign it to the developer again to test and change the status of the bug to "Reopen."
7. **Verified**: If the tester does not detect any issue in the bug after assigning it to the developer for retesting and if he senses that the bug has been fixed correctly, then he will mark the bug's status as "Verified."
8. **Closed:** When the bug is no longer available, then the tester changes the bug's status to "Closed."
9. **Duplicate:** If the developer discovers the bug-like any other bug or if the concept of the bug matches with any other bug, then the developer changes the bug's status to "Duplicate."
10. **Rejected:** If the developer does not consider the bug as a genuine bug, then he will mark it as "Rejected."
11. **Not a Bug:** If the bug does not affect the functionality of the application, then its status changes to "Not a Bug."
12. **Deferred:** If the developer senses that the bug is not of very critical priority and it can be fixed in the subsequent releases, he changes the bug's status as "Deferred."

Manual Testing Test Case


Header

Test Case name/ID:- Relese - Version - Application Name - Module
Test Case Type:- F.T.C | I.T.C | S.T.C
Requirement Number:-
Module:-
Servertiy:- Critical/Major/Minor
Status:-
Release:-
Version:-
Pre-Condition:-
Test Data:-
Summary:-

Body

Step No.	Description	Inputs	Expected Result	Actual Result	Status	Comments
•••	•••	•••	•••	•••	•••	•••
•••	•••	•••	•••	•••	•••	•••

Footer

Author:-	Reviewd By:-
Date:-	Approved By:-

Unit 2 – Automation Testing:

What is Automation Testing?

Automation testing is the process of testing the software application using an automation test tool to find the defects.

In this process, executing the test scripts and generating the results are performed automatically by test automation tools.

When to use Automation Testing?

Regression Testing

Repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the changes in the software being tested or in other related or unrelated software components. So, regression testing is best suitable for automated testing because of frequent code changes and it is beyond the human capacity to execute tests in a timely manner.

Load Testing

It is to verify that the system/application can handle the expected number of transactions and to verify the system/application’s behavior under both normal and peak load conditions. Automated testing is also the best way to complete the testing efficiently when it comes to load testing. It is best suited for automation testing.

Performance Testing

This type of testing determines or validates the speed, scalability, and/or stability characteristics of the system or application under test. Performance is concerned with achieving response times, throughput, and resource-utilization levels that meet the performance objectives for the project or product. It is best suited for automation testing.

Integration Testing

Integration Testing is the process of testing the interface between the two software units. Integration testing is done by multiple approaches such as Big Bang Approach, Top-Down Approach, Bottom-Up Approach, and the Hybrid Integration approach.

System Testing

Testing the fully integrated application to evaluate the system’s compliance with its specified requirements is called System Testing AKA End to End testing. Verifying the completed system to ensure that the application works as intended or not.

Unit Testing

Unit Testing is also called Module Testing or Component Testing. It is done to check whether the individual unit or module of the source code is working properly. It is done by the developers in the developer’s environment.

Acceptance Testing

It is also known as pre-production testing. This is done by the end-users along with the testers to validate the functionality of the application. After successful acceptance testing. Formal testing is conducted to determine whether an application is developed as per the requirement. It allows the customer to accept or reject the application. Types of acceptance testing are Alpha, Beta & Gamma.

Which tests cannot be automated?

Let us see which tests cannot be automated. Test which takes too much effort to automate are

1. Exploratory Testing
2. User interface testing
3. Adhoc Testing

When do you prefer Automation Testing over Manual Testing?

We prefer Manual Testing over Automation Testing in the following scenarios

1. It handles repetitive and time-consuming tasks
2. To do parallel testing
3. To do non-functional testing like load, performance, stress testing
4. Avoids human errors

Which Test Cases to Automate?

Test Cases to automate are as follows

1. Data-driven test cases
2. Test cases with higher complexity
3. Test case with many databases updates
4. The test execution rate is high
5. Smoke/Critical tests
6. Tests with several combinations
7. Graph test cases
8. Higher manual execution time

Which Test Cases are Not to be Automated?

Types of tests that need to be performed manually are as follows

1. Subjective Validation

2. New Functionalities
3. Strategic Development
4. User Experience
5. Complex Functionality
6. Quality Control
7. Low return on investment
8. Installation and setup testing

Automation Testing Vs. Manual Testing:

Automation Testing	Manual Testing
Automated testing is more reliable. It performs the same operation each time. It eliminates the risk of human errors.	Manual testing is less reliable. Due to human error, manual testing is not accurate all the time.
The initial investment in automation testing is higher. Investment is required for testing tools. In the long run, it is less expensive than manual. ROI is higher in the long run compared to Manual testing.	An initial investment in manual testing is less than automation. Investment is required for human resources. ROI is lower in the long run compared to Automation testing.
Automation testing is a practical option when we do regression testing.	Manual testing is a practical option where the test cases are not run repeatedly and only need to run once or twice.
Execution is done through software tools, so it is faster than manual testing and needs fewer human resources compared to manual testing.	Execution of test cases is time-consuming and needs more human resources
Exploratory testing is not possible	Exploratory testing is possible
Performance Testing like Load Testing, Stress Testing etc. is a practical option in automation testing.	Performance Testing is not a practical option in manual testing

Automation Testing	Manual Testing
It can be done in parallel and reduce test execution time.	It is not an easy task to execute test cases in parallel in manual testing. We need more human resources to do this and becomes more expensive.
Programming knowledge is a must in automation testing	Programming knowledge is not required to do manual testing.
Build verification testing (BVT) is highly recommended	Build verification testing (BVT) is not recommended
Human intervention is not much, so it is not effective to do User Interface testing.	It involves human intervention, so it is highly effective to do User Interface testing.

Automation Testing Tools

Some of the popular automation testing tools

- HP QTP (Quick Test Professional) / UFT (Unified Functional Testing)
- Selenium
- LoadRunner
- IBM Rational Functional Tester

Selenium:

Advent of Selenium

Jason Huggins, an engineer at ThoughtWorks, Chicago, found manual testing repetitive and boring. He developed a JavaScript program to automate the testing of a web application, called JavaScriptTestRunner.

Initially, the new invention was deployed by the employees at Thoughtworks. However, in 2004, it was renamed Selenium and was made open source. Since its inception, Selenium has been a powerful automation testing tool to test various web applications across different platforms.

What is Selenium?

Selenium is an open-source, automated testing tool used to test web applications across various browsers. Selenium can only test web applications, unfortunately, so desktop and mobile apps can’t be tested. However, other tools like Appium and HP’s QTP can be used to test software and mobile applications.

What makes Selenium Such a Widely Used Testing Tool?

1. Selenium is easy to use since it’s primarily developed in JavaScript
2. Selenium can test web applications against various browsers like Firefox, Chrome, Opera, and Safari
3. Tests can be coded in several programming languages like Java, Python, Perl, PHP, and Ruby
4. Selenium is platform-independent, meaning it can deploy on Windows, Linux, and Macintosh
5. Selenium can be integrated with tools like JUnit and TestNG for test management

Selenium Integrated Development Environment (IDE)

Developed by Shinya Kasatani in 2006, Selenium IDE is a browser extension for Firefox or Chrome that automates functionality. Typically, IDE records user interactions on the browser and exports them as a reusable script.

IDE was developed to speed up the creation of automation scripts. It’s a rapid prototyping tool and can be used by engineers with no programming knowledge whatsoever.

IDE ceased to exist in August 2017 when Firefox upgraded to a new Firefox 55 version, which no longer supported Selenium IDE. Applitools rewrote the old Selenium IDE and released a new version in 2019. The latest version came with several advancements.

Selenium IDE has a few shortcomings:

1. It does not support data-driven testing
2. It cannot perform database testing
3. It cannot provide a detailed test report
4. It cannot export to WebDriver scripts

Selenium Remote Control (RC)

Paul Hammant developed Selenium Remote Control (RC). Before we dive into RC, it’s important to know why RC came to be in the first place.

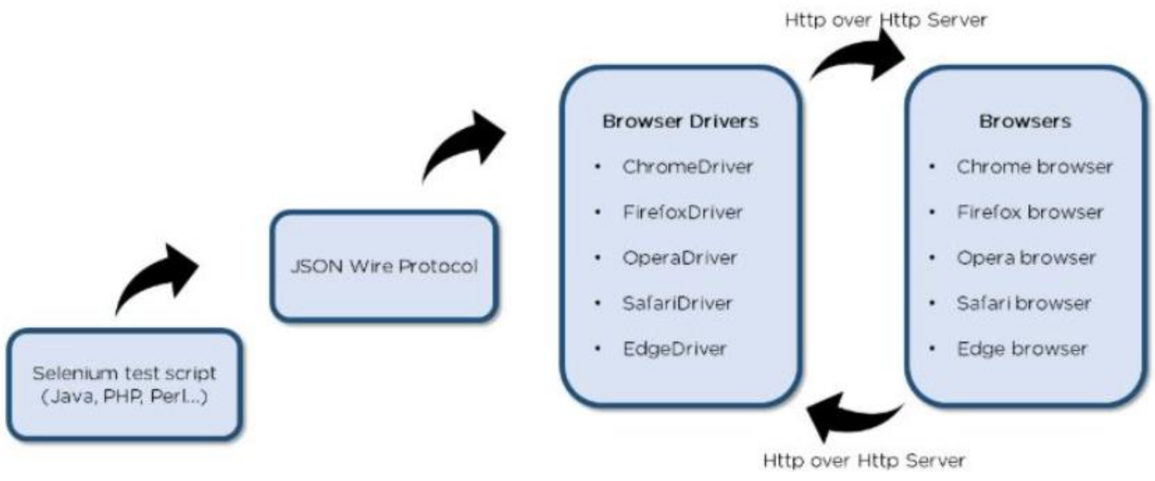
Initially, a tool called Selenium-Core was built. It was a set of JavaScript functions that interpreted and executed Selenese commands using the browser's built-in JavaScript interpreter. Selenium-Core was then injected into the web browser.

Selenium WebDriver

Developed by Simon Stewart in 2006, Selenium WebDriver was the first cross-platform testing framework that could configure and control browsers on the OS level. It served as a programming interface to create and run test cases.

Unlike Selenium RC, WebDriver doesn’t require a core engine like RC and interacts natively with browser applications. WebDriver also supports various programming languages like Python, Ruby, PHP, and Perl, among others, and can be integrated with frameworks like TestNG and JUnit for test management.

The architecture of Selenium WebDriver is simple and easy to understand:



- Selenium test script - Selenium test script is the test code written in any programming language be it Java, Perl, PHP, or Python that can be interpreted by the driver.
- JSON Wire Protocol - JSON Wire Protocol provides a transport mechanism to transfer data between a server and a client. JSON Wire Protocol serves as an industry standard for various web services.
- Browser drivers - Selenium uses drivers specific to each browser to establish a secure connection with the browser.
- Browsers - Selenium WebDriver supports various web browsers to test and run applications on.

Selenium Grid

Patrick Lightbody developed a grid with the primary objective of minimizing the test execution time. This was facilitated by distributing the test commands to different machines simultaneously. Selenium Grid allows the parallel execution of tests on different browsers and different operating systems. Grid is exceptionally flexible and integrates with other suite components for simultaneous execution.

Advantages of Selenium Testing:

Selenium automated testing comes with several benefits, such as:

1. Selenium has proven to be accurate with results thus making it extremely reliable
2. Since selenium is open-source, anybody willing to learn testing can begin at no cost
3. Selenium supports a broad spectrum of programming languages like Python, PHP, Perl, and Ruby
4. Selenium supports various browsers like Chrome, Firefox, and Opera, among others
5. Selenium is easy to implement and does not require the engineer to have in-depth knowledge of the tool
6. Selenium has plenty of re-usability and add-ons

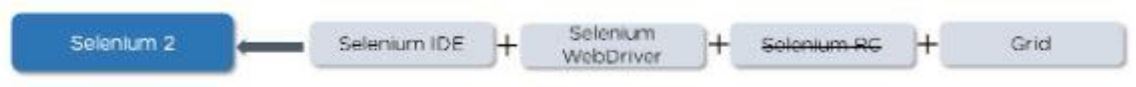
Limitations of Selenium Testing:

Selenium has a few shortcomings, which can include:

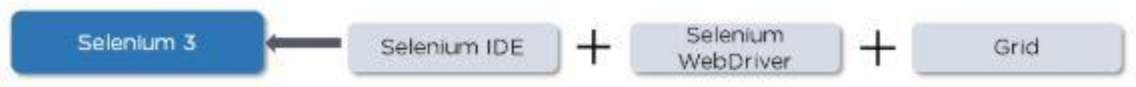
1. Since Selenium is open-source, it does not have a developer community and hence doesn’t have a reliable tech support
2. Selenium cannot test mobile or desktop applications
3. Selenium offers limited support for image testing
4. Selenium has limited support for test management. Selenium is often integrated with tools like JUnit and TestNG for this purpose
5. You may need knowledge of programming languages to use Selenium

What is the difference between Selenium 2.0 and Selenium 3.0?

Selenium 2.0 is a tool that makes the development of automated tests for web applications easier. It represents the merger of the original Selenium project with the WebDriver project. Selenium RC got deprecated since the merge, however, was used for backward compatibility



Selenium 3.0 is the extended version of Selenium 2.0. It is inherently backward compatible and does not involve Selenium RC. The new version came along with several bug fixes and increased stability.



What is the same-origin policy and how is it handled?

The same Origin policy is a feature adopted for security purposes. According to this policy, a web browser allows scripts from one webpage to access the contents of another webpage provided both the pages have the same origin. The origin refers to a combination of the URL scheme, hostname, and port number.

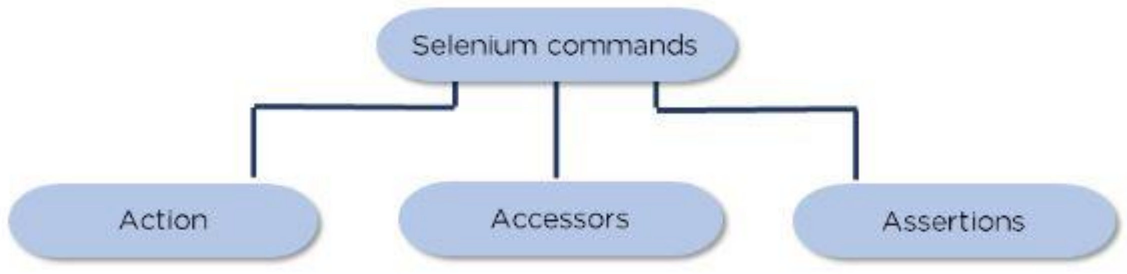
The same Origin Policy prevents a malicious script on one page to access sensitive data on another webpage.

Consider a JavaScript program used by google.com. This test application can access all Google domain pages like google.com/login, google.com/mail, etc. However, it cannot access pages from other domains like yahoo.com

Selenium RC was introduced to address this. The server acts as a client-configured HTTP proxy and "tricks" the browser into believing that Selenium Core and the web application being tested come from the same origin.

What is Selenese? How is it classified?

Selenese is the set of Selenium commands which are used to test your web application. The tester can test the broken links, the existence of some object on the UI, Ajax functionality, alerts, window, list options, and a lot more using Selenese.



Action: Commands which interact directly with the application

Accessors: Allow the user to store certain values to a user-defined variable

Assertions: Verifies the current state of the application with an expected state

What is the difference between Selenium IDE, Selenium RC, and WebDriver?

Feature	**Selenium IDE**	**Selenium RC**
Browser Compatibility	Selenium IDE comes as a Firefox plugin, thus it supports only Firefox	Selenium RC supports a varied range of versions of Mozilla Firefox, Google Chrome, Internet Explorer and Opera.
Record and Playback	Selenium IDE supports record and playback feature	Selenium RC doesn't supports record and playback feature.
Server Requirement	Selenium IDE doesn't require any server to be started before executing the test scripts	Selenium RC requires server to be started before executing the test scripts.
Architecture	Selenium IDE is a Javascript based framework	Selenium RC is a JavaScript based Framework.
Object Oriented	Selenium IDE is not an object oriented tool	Selenium RC is semi object oriented tool.
Dynamic Finders (for locating web elements on a webpage)	Selenium IDE doesn't support dynamic finders	Selenium RC doesn't support dynamic finders.

Feature	**Selenium IDE**	**Selenium RC**
Handling Alerts, Navigations, Dropdowns	Selenium IDE doesn't explicitly provides aids to handle alerts, navigations, dropdowns	Selenium RC doesn't explicitly provides aids to handle alerts, navigations, dropdowns.
WAP (iPhone/Android) Testing	Selenium IDE doesn't support testing of iPhone/Andriod applications	Selenium RC doesn't support testing of iPhone/Android applications.
Listener Support	Selenium IDE doesn't support listeners	Selenium RC doesn't support listeners.
Speed	Selenium IDE is fast as it is plugged in with the web-browser that launches the test. Thus, the IDE and browser communicates directly	Selenium RC is slower than WebDriver as it doesn't communicates directly with the browser; rather it sends selenese commands over to Selenium Core which in turn communicates with the browser.

When should I use Selenium IDE?

Selenium IDE is the simplest and easiest of all the tools within the Selenium Package. Its record and playback feature makes it exceptionally easy to learn with minimal acquaintance to any programming language. Selenium IDE is an ideal tool for a naïve user.

What is the difference between assert and verify commands?

Assert: Assert command checks whether the given condition is true or false. Let’s say we assert whether the given element is present on the web page or not. If the condition is true then the program control will execute the next test step but if the condition is false, the execution would stop and no further test would be executed.

Verify: Verify command also checks whether the given condition is true or false. Irrespective of the condition being true or false, the program execution doesn’t halt i.e., any failure during verification would not stop the execution and all the test steps would be executed.

UNIT 3 – SELENIUM IDE and WebDriver

Handling Form Elements:

- **Radio buttons:** To select a radio button, use the "click" function. For example, if the radio button has an ID of "option1", the code would be: `driver.findElement(By.id("option1")).click();`
- **Dropdown lists:** To select an option from a dropdown list, use the "Select" class. For example, if the dropdown list has an ID of "country", the code would be:

  ```
  Select dropdown = new
  Select(driver.findElement(By.id("country")));
  dropdown.selectByVisibleText("United States");
  ```

- **Buttons:** To click on a button, use the "click" function. For example, if the button has an ID of "submit", the code would be:

  ```
  driver.findElement(By.id("submit")).click();
  ```

- **File uploads:** To upload a file, use the "**sendKeys**" function and provide the file path as the argument. For example, if the file input has an ID of "**fileUpload**", the code would be:

  ```
  WebElement uploadElement =
  driver.findElement(By.id("fileUpload"));
  uploadElement.sendKeys("C:\path\to\file.txt");
  ```

Sample Code:

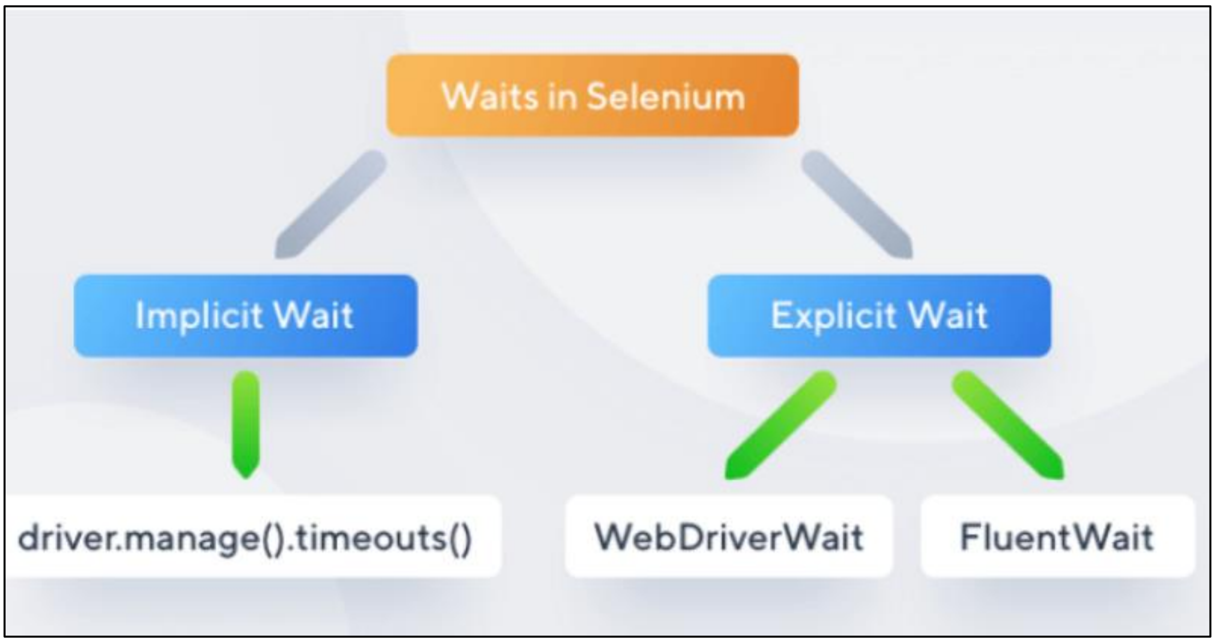
```
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.Select;

public class FormHandlingExample {

   public static void main(String[] args) {

    System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

     WebDriver driver = new ChromeDriver();

      driver.get("http://www.example.com/form");

WebElement usernameField = driver.findElement(By.id("username"));

      usernameField.sendKeys("myusername");

WebElement rememberMeCheckbox = driver.findElement(By.id("rememberMe"));

      rememberMeCheckbox.click();

WebElement option1RadioButton = driver.findElement(By.id("option1"));

      option1RadioButton.click();

WebElement submitButton = driver.findElement(By.id("submit"));

      submitButton.click();

driver.quit();

   }
```

Selenium Waits:

Waits in Selenium is one of the important pieces of code that executes a test case. It runs on certain commands called scripts that make a page load through it. Selenium Waits makes the pages less vigorous and reliable.

Another instance to understand Selenium Waits is by navigating web pages back and forth with the navigate() command. This navigate() method comes from WebDriver

Types of Selenium Waits:



Implicit waits:

The main function of implicit Wait is to tell the web driver to wait for some time before throwing a "No Such Element Exception". Its default setting is knocked at zero. Once the time is set, the driver automatically will wait for time defined by you before throwing the above-given exception.

Syntax:

```
driver.manage().timeouts().implicitlyWait(TimeOut,

TimeUnit.SECONDS);
```

Explicit waits:

Explicit Waits are also known as Dynamic Waits because it is highly specific conditioned. It is implemented by WebDriverWait class.

Syntax:

```
WebDriverWait wait=new

WebDriverWait(WebDriveReference,TimeOut);
```

The above syntax justifies an object of WebDriver Wait and is passed to the driver's preference, and the timeout is taken as a parameter.

Fluent wait:

Fluent Wait is quite like explicit Wait, it can perform wait for action for an element only when you are unaware of the time it might take to be clickable or visible. Pooling Frequency and Ignore Exceptions are two differential factors

Syntax:

```
  Wait<WebDriver> fluentWait = new FluentWait<WebDriver>(driver)

       .withTimeout(60, SECONDS)

       .pollingEvery(2, SECONDS)

       .ignoring(NoSuchElementException.class);


       WebElement foo = fluentWait.until(new Function<WebDriver,WebElement>)
```

Mention the types of navigation commands

driver.navigate().to("https://www.ebay.in/"); - Navigates to the provided URL

driver.navigate().refresh(); - This method refreshes the current page driver.navigate().forward(); - This method does the same operation as clicking on the Forward Button of any browser. It neither accepts nor returns anything.

driver.navigate().back(); - This method does the same operation as clicking on the Back Button of any browser. It neither accepts nor returns anything.

What is the major difference between driver.close() and driver.quit()?

driver.close() - This command closes the browser's current window. If multiple windows are open, the current window of focus will be closed.

driver.quit() - When quit() is called on the driver instance and there are one or more browser windows open, it closes all the open browser windows.

How to type text in an input box using Selenium?

sendKeys() is the method used to type text in input boxes

Consider the following example -

WebElement email = driver.findElement(By.id("email")); - Finds the "email" text using the ID locator

email.sendKeys("abcd.efgh@gmail.com"); - Enters text into the URL field

WebElement password = driver.findElement(By.id("Password")); - Finds the "password" text using the ID locator

password.sendKeys("abcdefgh123"); - Enters text into the password field

How to click on a hyperlink in Selenium?

driver.findElement(By.linkText("Today's deals")).click();

The command finds the element using link text and then clicks on that element, where after the user would be redirected to the corresponding page.

driver.findElement(By.partialLinkText("Service")).click();

The above command finds the element based on the substring of the link provided in the parenthesis and thus partialLinkText() finds the web element.

How to assert the title of a webpage?

Get the title of the webpage and store it in a variable

```
    String actualTitle = driver.getTitle();
```

Type in the expected title

```
   String expectedTitle = “abcdefgh";
```

Verify if both of them are equal

```
   if(actualTitle.equalsIgnoreCase(expectedTitle))

   System.out.println("Title Matched");

  else

  System.out.println("Title didn't match");
```

Alternatively,

```
   Assert.assertEquals(actualTitle, expectedTitle);
```

How to take screenshots in WebDriver?

TakeScreenshot interface can be used to take screenshots in WebDriver.

getScreenshotAs() method can be used to save the screenshot

```
File scrFile = ((TakeScreenshot)driver).getScreenshotAs(outputType.FILE);
```

Is there a way to type in a textbox without using sendKeys()?

Yes! Text can be entered into a textbox using JavaScriptExecutor

```
JavascriptExecutor jse = (JavascriptExecutor) driver;
```

```
jse.executeScript("document.getElementById(‘email').value=“abc.efg@xyz.com”);
```

Generation of Script: Selenium IDE provides a record and playback feature that allows users to record their interactions with a web application and generate test scripts automatically.

1. **Open Selenium IDE:** Open the Selenium IDE in the Firefox browser.
2. **Create a new test case:** Click on the "New Test Case" button to create a new test case.
3. **Record interactions:** Click on the "Record" button to start recording. Perform the interactions you want to test, such as entering text into text fields, clicking buttons, selecting options from dropdowns, and so on. All your interactions will be recorded automatically.
4. **Stop recording:** Click on the "Stop" button to stop recording.
5. **Export the script:** Click on the "Export" button to export the recorded test case as a script in the programming language JAVA
6. **Save the script:** Save the generated script to a file with a .java
7. **Run the script:** You can now use the generated script to run your automated tests on different browsers or platforms.

Selenium Locators:

Locating elements in Selenium WebDriver is performed with the help of findElement() and findElements() methods provided by WebDriver and WebElement class.

- **findElement()** returns a WebElement object based on specified search criteria or ends up throwing an exception if it does not find any element matching the search criteria.
- **findElements()** returns a list of WebElements matching the search criteria. If no elements are found, it returns an empty list**.**

Mention the types of Web locators.

Locator is a command that tells Selenium IDE which GUI elements (say Text Box, Buttons, Check Boxes, etc) it needs to operate on. Locators specify the area of action.

Locator by ID: It takes a string parameter which is a value of the ID attribute which returns the object to the findElement() method.

```
driver.findElement(By.id(“user”));
```

Locator by the link: If your targeted element is a link text then you can use the by.linkText locator to locate that element.

```
driver.findElement(By.linkText(“Today’s deals”)).click();
```

Locator by Partial link: The target link can be located using a portion of text in a link text element.

```
driver.findElement(By.linkText(“Service”)).click();
```

Locator by Name: The first element with the name attribute value matching the location will be returned.

```
driver.findElement(By.name(“books”).click());
```

Locator by TagName: Locates all the elements with the matching tag name

```
driver.findElement(By.tagName(“button”).click());
```

Locator by classname: This finds elements based on the value of the CLASS attribute. If an element has many classes then this will match against each of them.

```
driver.findElement(By.className(“inputtext”));
```

Locator by XPath: It takes a parameter of String which is an XPATHEXPRESSION and it returns an object to findElement() method.

```
driver.findElement(By.xpath(“//span[contains(text(),’an account’)]”)).getText();
```

Locator by CSS Selector: Locates elements based on the driver's underlying CSS selector engine.

```
driver.findElement(By.cssSelector("input#email")).sendKeys("myemail@email.com");
```

UNIT 4 – Selenium WebDriver

How to select a value in a dropdown?

The value in the dropdown can be selected using WebDriver's Select class.

Syntax:

selectByValue:
*Select selectByValue = **new** Select(driver.findElement(By.id("SelectID_One")));*
selectByValue.selectByValue("greenvalue");

selectByVisibleText:

*Select selectByVisibleText = **new** Select (driver.findElement(By.id("SelectID_Two")));*

selectByVisibleText.selectByVisibleText("Lime");

selectByIndex:

*Select selectByIndex = **new** Select(driver.findElement(By.id("SelectID_Three")));*

selectByIndex.selectByIndex(2);

How to handle frame in WebDriver?

An inline frame acronym as iframe is used to insert another document within the current HTML document or simply a web page into a web page by enabling nesting.

Select iframe by id

driver.switchTo().frame("ID of the frame");

Locating iframe using tagName

driver.switchTo().frame(driver.findElements(By.tagName("iframe").get(0));

Locating iframe using index

frame(index)

driver.switchTo().frame(0);

frame(Name of Frame)

driver.switchTo().frame("name of the frame");

frame(WebElement element)

Select Parent Window

driver.switchTo().defaultContent();

Can Selenium handle Windows-based pop-ups?

Selenium is an automation testing tool that supports only web application testing. Therefore, windows pop-ups cannot be handled using Selenium.

How can we handle web-based pop-ups?

WebDriver offers users a very efficient way to handle these pop-ups using the Alert interface. There are four methods that we would be using along with the Alert interface.

- void dismiss() – The dismiss() method clicks on the "Cancel" button as soon as the pop-up window appears.
- void accept() – The accept() method clicks on the "Ok" button as soon as the pop-up window appears.
- String getText() – The getText() method returns the text displayed on the alert box.
- void sendKeys(String stringToSend) – The sendKeys() method enters the specified string pattern into the alert box.

How can we handle windows based pop up?

Selenium is an automation testing tool that supports only web application testing, that means, it doesn't support testing of windows based applications. However Selenium alone can't help the situation but along with some third-party intervention, this problem can be overcome. There are several third-party tools available for handling window-based pop-ups along with the selenium like AutoIT, Robot class etc.

How to assert the title of the web page?

```
assertTrue(“The title of the window is
incorrect.”,driver.getTitle().equals(“Title of the page”));
```

How to retrieve the CSS properties of an element?

The values of the CSS properties can be retrieved using a get() method:

Syntax:

```
driver.findElement(By.id(“id“)).getCssValue(“name of css
attribute”);
driver.findElement(By.id(“id“)).getCssValue(“font-size”);
```

WebDriver Commands:

In Selenium WebDriver, we have an entirely different set of commands for performing different operations. Since we are using Selenium WebDriver with Java, commands are simply **methods** written in Java language.

1. Fetching a web page

There are two methods to fetch a web page:

Using Get method - **driver.get("www.javatpoint.com")**

Using Navigate method - **driver.navigate().to("https://javatpoint.com/selenium-tutorial");**

2. Locating forms and sending user inputs

    driver.findElement(By.id("lst-ib")).sendKeys("javatpoint tutorials");

3. Clearing User inputs

The clear() method is used to clear the user inputs from the text box.

driver.findElement(By.name("q")).clear();

4. Fetching data over any web element

Sometimes we need to fetch the text written over a web element for performing some assertions and debugging. We use getText() method to fetch data written over any web element.

```
driver.findElement(By.id("element567")).getText();
```

5. Performing Click event

The click() method is used to perform click operations on any web element.

```
driver.findElement(By.id("btnK")).click();
```

6. Navigating backward in browser history

```
driver.navigate().back();
```

7. Navigating forward in browser history

```
driver.navigate().forward();
```

8. Refresh/ Reload a web page

```
driver.navigate().refresh();
```

9. Closing Browser

```
driver.close();
```

10. Closing Browser and other all other windows associated with the driver

```
driver.quit();
```

11. Moving between Windows

```
driver.switchTo().window("windowName");
```

13. Moving between Frames

```
driver.switchTo().frame("frameName");
```

14. Drag and Drop

Drag and Drop operation is performed using the Action class.

```
1. WebElement element = driver.findElement(By.name("source"));
2. WebElement target = driver.findElement(By.name("target"));
3. (new Actions(driver)).dragAndDrop(element, target).perform();
```

Selenium Exceptions:

Selenium is a web automation framework that allows testing applications against different browsers. During automation in Selenium, the testing team must handle multiple exceptions.

Exceptions are faults or disruptions that occur during the execution of a program/application. Exception handling is crucial for maintaining the natural or normal flow of the application.

Selenium exceptions can be broadly categorized into two types: Checked and Unchecked Exceptions.

Checked exceptions are handled during the coding process itself. Unchecked exceptions occur during run-time and can have a much greater impact on the application flow. We have compiled some of the most common selenium exceptions along with the various ways to handle them.

Most Common Selenium Exceptions

- NoSuchWindowException
- NoSuchFrameException
- NoSuchElementException
- NoAlertPresentException
- InvalidSelectorException
- TimeoutException
- ElementNotVisibleException
- ElementNotSelectableException
- NoSuchSessionException
- StaleElementReferenceException

1. NoSuchWindowException

One of the most frequent exceptions in Selenium Webdriver, NoSuchWindowException occurs if the current list of windows is not updated. The previous window does not exist, and you can't switch to it.

To handle this exception, use webdriver methods called "driver.getWindowHandles()."

2. NoSuchFrameException

Similar to NoSuchWindowException, the NoSuchFrameException occurs when switching between multiple frames is not possible.

The solution used for handling NoSuchWindowException must ideally work for this exception too.

3. NoSuchElementException

Happens when the locators are unable to find or access elements on the web page or application. Ideally, the exception occurs due to the use of incorrect element locators in the findElement(By, by) method.

To handle this exception, use the wait command. Use Try/Catch to ensure that the program flow is interrupted if the wait command doesn't help.

4. NoAlertPresentException

Happens when the user is trying to you switch to an alert which is not present. In simple terms, it means that the test is too quick and is trying to find an alert that has not yet been opened by the browser.

To handle or simply avoid this exception, use explicit or fluent wait in all events where an alert is expected.

5. InvalidSelectorException

This exception occurs due to an incorrect selector. More often than not, the selector syntax is wrong.

To avoid this exception, first, check the locator syntax. If it is incorrect, make sure the locator is syntactically correct.

6. TimeoutException

This exception is thrown if the command did not execute or complete within wait time. As already mentioned, waits are used to avoid NoSuchElementException. However, TimeoutException will be thrown after the page components fail to load within wait time.

Avoiding this exception is simple. All one needs to do is to calculate the average page load time and adjust the wait time accordingly.

7. ElementNotVisibleException

This exception occurs when the WebDriver tries to find an element that is hidden or invisible. To handle this exception, it is essential that the exact reason is identified, which can be due to nested web elements or overlapping web elements.

In the first case, you have to locate and correct the specific element. In the second case, use explicit wait.

8. ElementNotSelectableException

This exception belongs to the same class as InvalidElementStateException. In such exceptions, the element is present on the web page, but the element cannot be selected.

To handle this exception, the wait command must be used.

9. NoSuchSessionException

As the name suggests, the exception is thrown if a method is called post the browser is closed. Other reasons for this exception include a browser crash.

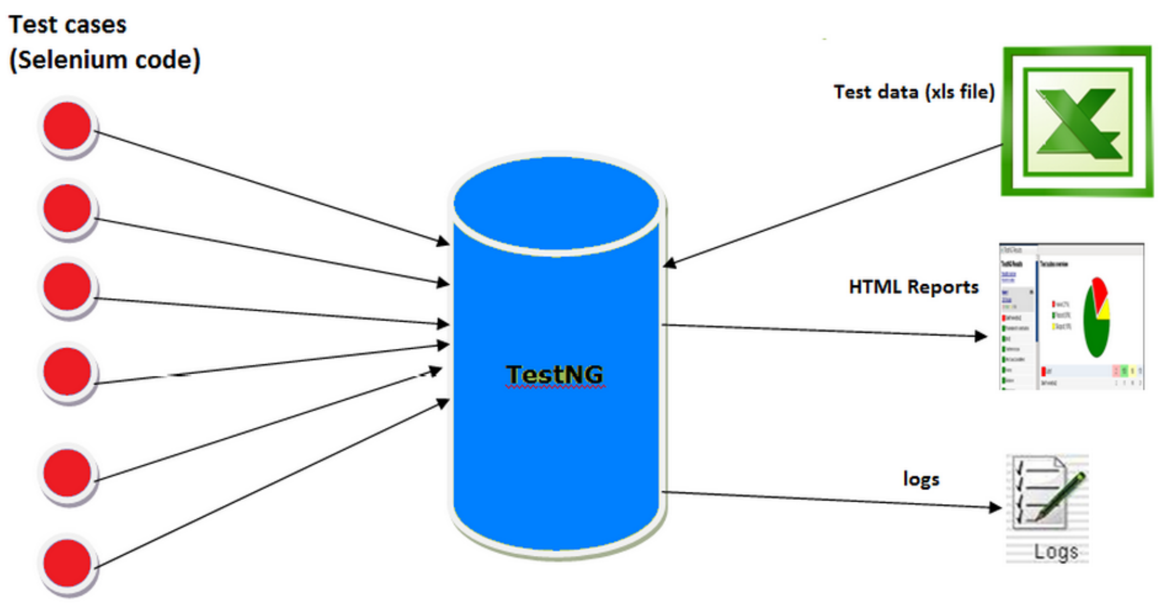
To avoid this handle, ensure that your browser is updated and stable.

10. StaleElementReferenceException

This exception occurs when the web element is no longer part of the web page. The element may have been part of the source code, but it is no longer part of the window. There can be multiple reasons for this exception. It can occur either from a switch to a different page, the element is no longer part of DOM or due to a frame/window switch

To handle this exception, you can either use Dynamic Xpath for handling DOM operations or try to use the Page Factory Model or try to access the element in loops before throwing the exception.

UNIT 6 - Testing:



Maven Configuration:

- Maven is a build automation tool that is widely used in Java projects.
- It simplifies the process of building, managing, and deploying Java applications by providing a consistent and standardized approach to project configuration and management.
- The configuration of a Maven project is done using a Project Object Model (POM) file, which is an XML file that describes the project and its dependencies.

POM FILE:

- The POM file contains information about the project's name, version, dependencies, plugins, and other relevant information
- Some of the key elements in a Maven POM file:
- Group ID: This is a unique identifier for the project's group. It usually follows a reverse domain name convention, such as com.example.project.
- Artifact ID: This is a unique identifier for the project's artifact. It represents the project's name and is used to generate the name of the output file, such as the JAR file.
- Version: This specifies the version number of the project. It is used to distinguish different versions of the same project.
- Dependencies: This section lists the dependencies required by the project. Each dependency includes its own group ID, artifact ID, and version number.
- Plugins: This section lists the plugins used by the project. Plugins are used to extend the functionality of Maven, such as compiling code, running tests, and generating reports.

Commands in MAVEN:

- **Build:** This section describes the build process for the project. It includes information about the source code directory, output directory, and other relevant information. Here are some commonly used Maven commands:
- **mvn clean:** Deletes the target directory and removes any generated files.
- **mvn compile:** Compiles the source code and generates the class files.

- **mvn test:** Runs the tests in the project.
- **mvn package:** Generates the JAR file or other output formats specified in the POM file.
- **mvn install:** Installs the project's artifacts in the local repository for use by other projects.
- Overall, Maven provides a standardized and efficient way to manage Java projects and their dependencies.

Sample Code:

```
Add the Testng plugin to the pom.xml file
<build>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-surefire-   plugin</artifactId>
         <version>3.0.0-M5</version>
         <dependencies>
            <dependency>
<groupId>org.testng</groupId>
               <artifactId>testng</artifactId>
               <version>7.4.0</version>
               <scope>test</scope>
            </dependency>
         </dependencies>
      </plugin>
   </plugins>
</build>
Create a TestNG test class in your project for example
import org.testng.annotations.Test;
public class MyTestNGTest {
   @Test
   public void testMethod() {
      // Your test code here
```

TestNg:

- **TestNG** is a testing framework for Java applications that provides a powerful and flexible way to write and run tests. It was developed by Cédric Beust as a replacement for JUnit 3.x and has become one of the most popular testing frameworks for Java.
- TestNG stands for Test Next Generation, and it is designed to address some of the shortcomings of JUnit and provide additional features and capabilities
- **Easy to Use:** TestNG is easy to use and has a simple syntax, making it easy for developers to write tests.
- **Flexibility**: TestNG provides a wide range of annotations that can be used to control the test execution flow, making it more flexible and powerful than JUnit.
- **Parallel Execution**: TestNG provides the ability to execute tests in parallel, which can greatly **improve** the speed of test execution.
- **Reporting**: TestNG provides rich reporting capabilities, which can help developers and testers easily analyze and debug test failures.
- **Data-Driven Testing**: TestNG provides support for data-driven testing, which can greatly reduce the amount of code required to write tests.
- **TestNG Integrations**: TestNG integrates with a wide range of tools and frameworks, including Eclipse, Maven, Ant, Jenkins, and Selenium. This makes it easy to integrate TestNG into existing workflows and toolchains.
- **@Test**: This annotation is used to mark a method as a test method. It can be used to specify the expected exception, timeout, and invocation count.
- **@BeforeSuite**: This annotation is used to mark a method that should be run before all the tests in the suite.
- **@AfterSuite**: This annotation is used to mark a method that should be run after all the tests in the suite.
- **@BeforeTest**: This annotation is used to mark a method that should be run before all the tests in a specific test tag.

- **@AfterTest**: This annotation is used to mark a method that should be run after all the tests in a specific test tag.
- **@BeforeClass**: This annotation is used to mark a method that should be run before all the test methods in a class.
- **@AfterClass**: This annotation is used to mark a method that should be run after all the test methods in a class.
- **@BeforeMethod**: This annotation is used to mark a method that should be run before each test method.
- **@AfterMethod**: This annotation is used to mark a method that should be run after each test method.
- These annotations provide a powerful way to control the test execution flow, making it more flexible and powerful than JUnit.

Testing Assertion:

- TestNG provides a set of assertions that can be used to verify the expected results of a test case.
- These assertions are similar to the assertions provided by JUnit but are more powerful and flexible.
- Some of the most commonly used assertions are:
- assertEquals(expected, actual) - Tests that two values are equal
- assertNotEquals(expected, actual) - Tests that two values are not equal
- assertTrue(condition) - Tests that a condition is true
- assertFalse(condition) - Tests that a condition is false
- assertNull(object) - Tests that an object is null
- assertNotNull(object) - Tests that an object is not null
- assertSame(expected, actual) - Tests that two objects refer to the same object instance

- assertNotSame(expected, actual) - Tests that two objects do not refer to the same object instance
- assertThat(actual, matcher) - Tests that a value matches a Hamcrest matcher

Sample Code:

```
@Test
public void testMethod() {
 String str1 = "Hello";
 String str2 = "World";
 int num1 = 5;
 int num2 = 10;
assertEquals("Hello", str1);
 assertNotEquals(str1, str2);
 assertTrue(num1 < num2);
 assertNull(null);
 assertNotNull(str1);
 assertSame(str1, str1);
 assertNotSame(str1, str2);
 assertThat(str1, containsString("He"));
}
```

Testing Dependency Groups:

- TestNG allows you to specify dependencies between test methods and groups.
- This means that you can specify that a test method should only be run if another test method has passed or failed, or if it belongs to a specific group.
- This makes it easier to manage complex test suites and ensures that tests are executed in the correct order.
- TestNG provides the ability to run test cases using an XML configuration file.

- This allows you to configure the test suite and specify which tests to run, as well as any dependencies or groups.
- This XML configuration file can be executed using the testng.xml file name and TestNG will run all the specified test cases.

How to run the test script in TestNG?

You can run the test script in TestNG by clicking right-click on the TestNG class, clicking on "Run As" and then select "TestNG test".

What are the annotations used in the TestNG?

The following are the annotations used in the TestNG are:

- **Precondition annotations**
  Precondition annotations are executed before the execution of test methods The Precondition annotations are @BeforeSuite, @BeforeClass, @BeforeTest, @BeforeMethod.
- **Test annotation**
  Test annotation is specified before the definition of the test method. It is specified as @Test.
- **Postcondition annotations**
  The postcondition annotations are executed after the execution of all the test methods. The postcondition annotation can be @AfterSuite, @AfterClass, @AfterTest, @AfterMethod.

What is the sequence of execution of all the annotations in TestNG?

The sequence of execution of all the annotations in TestNG is given below:

- @BeforeSuite
- @BeforeTest
- @BeforeClass
- @BeforeMethod
- @Test

- @AfterSuite
- @AfterTest
- @AfterClass
- @AfterMethod

How to set the priorities in TestNG?

If we do not prioritize the test methods, then the test methods are selected alphabetically and executed. If we want the test methods to be executed in the sequence we want, then we need to provide the priority along with the @Test annotation.

Define grouping in TestNG?

The group is an attribute in TestNG that allows you to execute the multiple test cases. For example, if we have 100 test cases of it_department and 10 test cases of hr_department, and if you want to run all the test cases of it_department together in a single suite, this can be possible only through the grouping.

What is dependency in TestNG?

When we want to run the test cases in a specific order, then we use the concept of dependency in TestNG.

Two types of dependency attributes used in TestNG:

- **dependsOnMethods**
  The dependsOnMethods attribute tells the TestNG on which methods this test will be dependent on, so that those methods will be executed before this test method.
- **dependsOnGroups**
  It is similar to the dependsOnMethods attribute. It allows the test methods to depend on the group of test methods. It executes the group of test methods before the dependent test method.

What is timeOut in TestNG?

While running test cases, there can be a case when some test cases take much more time than expected. In such a case, we can mark the test case as a failed test case by using timeOut.

TimeOut in TestNG allows you to configure the time period to wait for a test to get completely executed. It can be configured in two levels:

- **At the suit level:** It will be available to all the test methods.
- **At each method level:** It will be available to a particular test method.

The timeOut attribute can be specified as shown below:

```
@Test( timeOut = 700)
```

The above @Test annotation tells that the test method will be given 700 ms to complete its execution otherwise it will be marked as a failed test case.

What is invocationCount in TestNG?

An invocation count in TestNG is the number of times that we want to execute the same test.

What is the importance of the testng.xml file?

The testng.xml file is important because of the following reasons:

- It defines the order of the execution of all the test cases.
- It allows you to group the test cases and can be executed as per the requirements.
- It executes the selected test cases.
- In TestNG, listeners can be implemented at the suite level.
- It allows you to integrate the TestNG framework with tools such as Jenkins.

How to pass the parameter in the test case through the testng.xml file?

We can also pass the value to test methods at runtime, we can achieve this by sending parameter values through the testng.xml file. We can use the **@Parameter** annotation:

```
@Parameter("param-name");
```

How can we disable the test case from running?

We can disable the test case from running by using the enabled attribute. We can assign the false value to the enabled attribute, in this way, we can disable the test case from running.

What is the difference between soft assertion and hard assertion?

Soft Assertion: In the case of Soft Assertion, if TestNG gets an error during @Test, it will throw an exception when an assertion fails and continues with the next statement after the assert statement.

Hard Assertion: In the case of Hard Assertion, if TestNG gets an error during @Test, it will throw an AssertException immediately when an assertion fails and stops execution after the assert statement.

What is the use of @Listener annotation in TestNG?

TestNG provides different kinds of listeners which can perform different actions whenever the event is triggered. The most widely used listener in TestNG is ITestListener interface. The ITestListener interface contains methods such as onTestSuccess, onTestfailure, onTestSkipped, etc.

Following are the scenarios that can be made:

- If the test case is failed, then what action should be performed by the listener.
- If the test case is passed, then what action should be performed by the listener.
- If the test case is skipped, then what action should be performed by the listener.

What is the use of @Factory annotation?

The @Factory annotation is useful when we want to run multiple test cases through a single test class. It is mainly used for the dynamic execution of test cases.

What is the difference between @Factory and @DataProvider annotation?

@DataProvider: It is an annotation used by TestNG to execute the test method multiple numbers of times based on the data provided by the DataProvider.

@Factory: It is an annotation used by the TestNG to execute the test methods present in the same test class using different instances of the respective class.

UNIT 7 – Testing Frameworks

Linear Scripting Framework

Linear Scripting Framework is a basic level test automation framework that is implemented in the form of "Record and Playback" and is executed linearly. Also known as the "Record and Playback" framework, this framework is used to record and playback audio. This sort of framework is used to test apps that are of a small size.

In this testing method, the preparation and execution of test scripts are done on an individual basis for each test case.

The testers record each test step, such as browsing, navigation, user inputs, and imposing checkpoints. The testers then play the scripts back and use them to carry out the tests.

The following are some of the **benefits** of using the Linear Scripting Automation Framework:

- It is possible to develop test scripts (record and playback) without spending a lot of time preparing ahead.
- It is not necessary to be familiar with coding.
- A simple method for quickly generating test scripts

The following are some of the **disadvantages** of the Linear Scripting Automation Framework:

- Because of auto-generated scripts, there is a lack of reusability.
- We are unable to execute with numerous data sets since the data has been hardcoded.
- Finally, the maintenance cost is substantial — even minor adjustments need a significant amount of work.

Modular Testing Framework

In the modular testing framework, testers construct test scripts module by module, breaking down the whole application under test into smaller, independent tests. Modular Testing Framework:

The testers break down the program into many parts and develop test scripts for each module independently, to put it another way. Combining these separate test scripts to create bigger ones by using a master script to produce the desired situations in each case. This master script is used to call the separate modules to perform end-to-end test scenarios from start to finish.

The primary rationale for using this framework is to provide an abstraction layer that protects the master module from modifications made in individual tests, which may be dangerous. In this framework, testers create function libraries that may be called upon whenever necessary. This is referred to as a modularity framework or a module-based framework in certain circles.

The following are the advantages of using a modular testing framework:

- Because the whole program has been broken down into various modules, it has been made more scalable and simpler to maintain.
- Can develop test scripts on their own initiative.
- When a change is made in one module, it has little or no influence on the other modules.

The following are the disadvantages of the Modular Testing Framework:

- First, it takes more time to examine the test cases and find reusable processes than it does to write them.
- Numerous data sets are not allowed because the test scripts include hard-coded data.
- Setting up the framework necessitates the use of coding abilities.

3. Library Architecture Testing Framework

Also known as "**Structured Scripting**" or "**Functional Decomposition**," the Library Architecture Testing Framework is a framework for testing library architecture. It has some extra benefits since it is built on a modular foundation.

Like splitting the program under test into modules in the modular testing framework, we identify the common tasks and combine them into functions in the functional testing framework (Figure 1). Once the functions have been grouped, the groupings will be stored in a library of sorts. Then, the test scripts re-use these libraries to generate new test cases.

The following are some of the **benefits** of using a Library Architecture Testing Framework:

- The script's upkeep is straightforward.
- Scalability is straightforward.
- The functions library is reusable, and you can re-use it.

The following are the **disadvantages** of using a Library Architecture Testing Framework:

- Coding skills are necessary.
- Preparing test scripts takes more time than usual.
- The scripts include a defined set of test data that has been hardcoded into them.

Data-driven Framework

The data-driven test automation framework is concerned with separating the logic of the test scripts from the data utilized in the tests themselves. It enables us to generate test automation scripts by giving the script generator multiple sets of test data.

It is necessary to save the test data set in external files or resources such as Microsoft Excel sheets, Microsoft Access tables (MS Access database), SQL database (SQL database), XML files, etc. The test scripts establish connections with other services to get test data. We were able to make the test scripts function quickly and simply correctly for various kinds of test data by using this framework. When compared to the module-based architecture, this framework lowers the number of test scripts by a substantial amount. Because of the use of reusable tests, this framework provides increased test coverage while also providing more flexibility in the execution of tests just when necessary and by altering only the input test data.

It is dependable in that it does not affect tests when the test data is changed, but it has its own limitations, such as the need for testers who work on this framework to have hands-on programming experience in order to build test scripts.

The following are some of the **advantages** of a data-driven framework:

- It is compatible with a variety of data sets.
- Making changes to the test scripts will have no effect on the test data.
- There is no need to hardcode test data.
- It saves time by running a greater number of tests.

A data-driven framework has the following **disadvantages**:

- Coding abilities are required.
- More time is required for the setup of the framework and test data.
- Framework design requires the expertise of professional automation testers.

5. **Keyword Driven Testing Framework**

Keyword Driven Testing Framework (also known as table-driven testing or action word-based testing) is a kind of testing framework that uses keywords to drive the testing process. We utilize a table structure in keyword-driven testing to declare keywords or action words for each function or method we want to test before executing it. Then, it runs automation test scripts based on the keywords given in the Excel sheet and reports the results.

Testers with no programming expertise can work on test automation scripts using this framework, which allows them to work with keywords to construct whatever test automation script they choose.

The code to read keywords from an external Excel sheet and call the appropriate action specified in the sheet is included inside the main class. Keyword-driven testing is like data-driven testing in that keywords drive it.

Even though working on this framework does not need much programming knowledge, the initial setup (i.e., putting the framework into operation) necessitates more competence.

The following are some of the **advantages** of keyword-driven frameworks:

- First, it is not necessary to be an expert in order to develop test scripts.
- It is possible to reuse the code in this situation. For example, we may direct the separate scripts to the same Keyword by using a variable.
- Even though the application changes, test scripts remain unchanged.
- You may create tests prior to the development of the application.
- Test scripts may be used independently of the application under test by making just minor changes.
- It is not reliant on test tools.

Disadvantages of Keyword-Driven Frameworks include the following:

- It will take more time to design since the initial cost will be expensive.
- Employees with strong test automation abilities are required.

6. **Hybrid Driven Testing Framework:**

A hybrid-driven testing framework is a mix of two or more of the frameworks discussed above. It is a kind of test automation framework. It tries to make use of the strengths and advantages of other frameworks for the specific test environment that it handles. In the present market, most teams are constructing this hybrid-driven structure.

7. Behavior Driven Development Testing Framework

This Testing Framework for Behavior Driven Development aims to develop a platform that enables everyone (including Business Analysts, Developers, and Testers, among others) to participate actively. More communication between the Development and Testing Teams is required. It does not need the knowledge of a programming language on the part of the consumers. In order to construct test requirements, we employ non-technical, everyday language. **JBehave**, **Cucumber**, and other similar tools are available on the market for Behavior Driven Development.

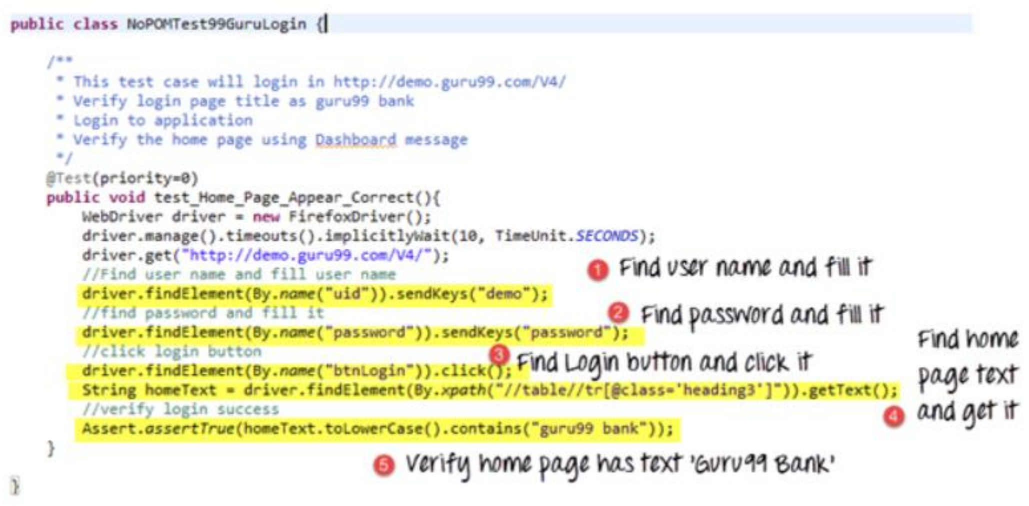
What is Page Object Model?

Page Object Model (POM) is a design pattern, popularly used in test automation that creates Object Repository for web UI elements. The advantage of the model is that it reduces code duplication and improves test maintenance.
Under this model, for each web page in the application, there should be a corresponding Page Class. This Page class will identify the WebElements of that web page and also contains Page methods which perform operations on those WebElements. Name of these methods should be

given as per the task they are performing, i.e., if a loader is waiting for the payment gateway to appear, POM method name can be waitForPaymentScreenDisplay().

Why Page Object Model?

Starting an UI Automation in Selenium WebDriver is NOT a tough task. You just need to find elements, perform operations on it.



```
public class NoPOMTest99GuruLogin {

    /**
     * This test case will login in http://demo.guru99.com/V4/
     * Verify login page title as guru99 bank
     * Login to application
     * Verify the home page using Dashboard message
     */
    @Test(priority=0)
    public void test_Home_Page_Appear_Correct(){
        WebDriver driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("http://demo.guru99.com/V4/");
        //Find user name and fill user name
        driver.findElement(By.name("uid")).sendKeys("demo");
        //find password and fill it
        driver.findElement(By.name("password")).sendKeys("password");
        //click login button
        driver.findElement(By.name("btnLogin")).click();
        String homeText = driver.findElement(By.xpath("//table//tr[@class='heading3']")).getText();
        //verify login success
        Assert.assertTrue(homeText.toLowerCase().contains("guru99 bank"));
    }
}
```

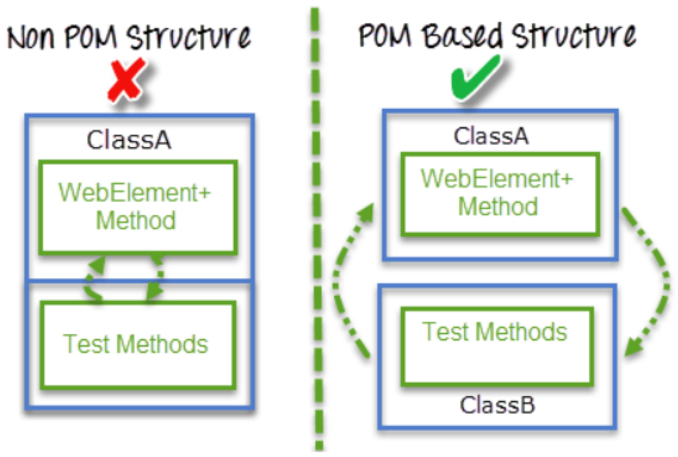
1 Find user name and fill it
2 Find password and fill it
3 Find Login button and click it
4 Find home page text and get it
5 Verify home page has text 'Guru99 Bank'

This is a small script. Script maintenance looks easy. But with time test suite will grow. As you add more and more lines to your code, things become tough.

The chief problem with script maintenance is that if 10 different scripts are using the same page element, with any change in that element, you need to change all 10 scripts. This is time consuming and error prone.

A better approach to script maintenance is to create a separate class file which would find web elements, fill them or verify them. This class can be reused in all the scripts using that element. In future, if there is a change in the web element, we need to make the change in just 1 class file and not 10 different scripts.

This approach is called Page Object Model in Selenium. It helps make the code more readable, maintainable, and reusable.



Advantages of POM

1. Page Object Design Pattern says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
2. The Second benefit is the object repository is independent of test cases, so we can use the same object repository for a different purpose with different tools. For example, we can integrate Page Object Model in Selenium with TestNG/JUnit for functional Testing and at the same time with JBehave/Cucumber for acceptance testing.
3. Code becomes less optimized because of the reusable page methods in the POM classes.
4. Methods get more realistic names that can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like ‘gotoHomePage()’.

UNIT 8 – Reports in Selenium

What are Test Reporting Tools?

Test Reporting tools help outline the key activities carried out during the test life cycle for any release. It also provides useful insights about a product that organizations can leverage to improve software quality.

It helps identify some important aspects of testing, such as:

- Scope of testing
- Tests that were performed

- Defects that were discovered

Why are test reporting tools required?

Test reporting tools can be very beneficial for organizations for the following reasons.

- **Detecting bugs early:** A good testing tool will help identify critical bugs very early in the cycle. This avoids having to manually look through the code to find bugs and then report them to the developer.
- **Makes analysis easier:** Test reporting tools generate reports that have information about errors, tests executed, test scenarios, and more. This provides valuable insight as to how good or bad of a shape the project is in and how it aligns with the business requirements.
- **Helps track the progress of a product:** Test reporting tools provide the necessary information to stakeholders about the product quality and help them keep track of a product and whether it is ready for release.

How to pick the right test automation reporting tool?

Since the quality of testing plays a huge role in deciding the roadmap of a project, it is vital to pick the best reporting tool for selenium. Here is how you can decide which is the right reporting tool by checking:

- **Functionality and features:** Do the Selenium Test Reporting Tool provide comprehensive and relevant results that can be useful to the team
- **Ease of use and understanding**
- **Usability:** Integration flexibility with various automation testing tools
- **Cost-effectiveness:** A cost-effective solution that caters to various features, capabilities, and use cases that you have

Top 3 Reporting Tools for Selenium

While Selenium has several reporting tools, here are some of the best automation reporting tools for Selenium:

- TestNG Reporter Log
- JUnit Reporter Log

- Extent Reports

TestNG Reporter Log

TestNG framework has an in-built reporting tool that comes with a default feature that generates reports in the form of an HTML file.

JUnit Reporter Log

JUnit is a popular framework that offers in-built reporting for the Selenium Tests.

Advantages of JUnit:

- It is an open-source framework
- It is simple and executes in lesser time
- They are easily accessible and require minimal coding

Limitations of JUnit:

- It works only with Java and JUnit
- It provides local reports only
- Some code implementation is required to make it function properly.

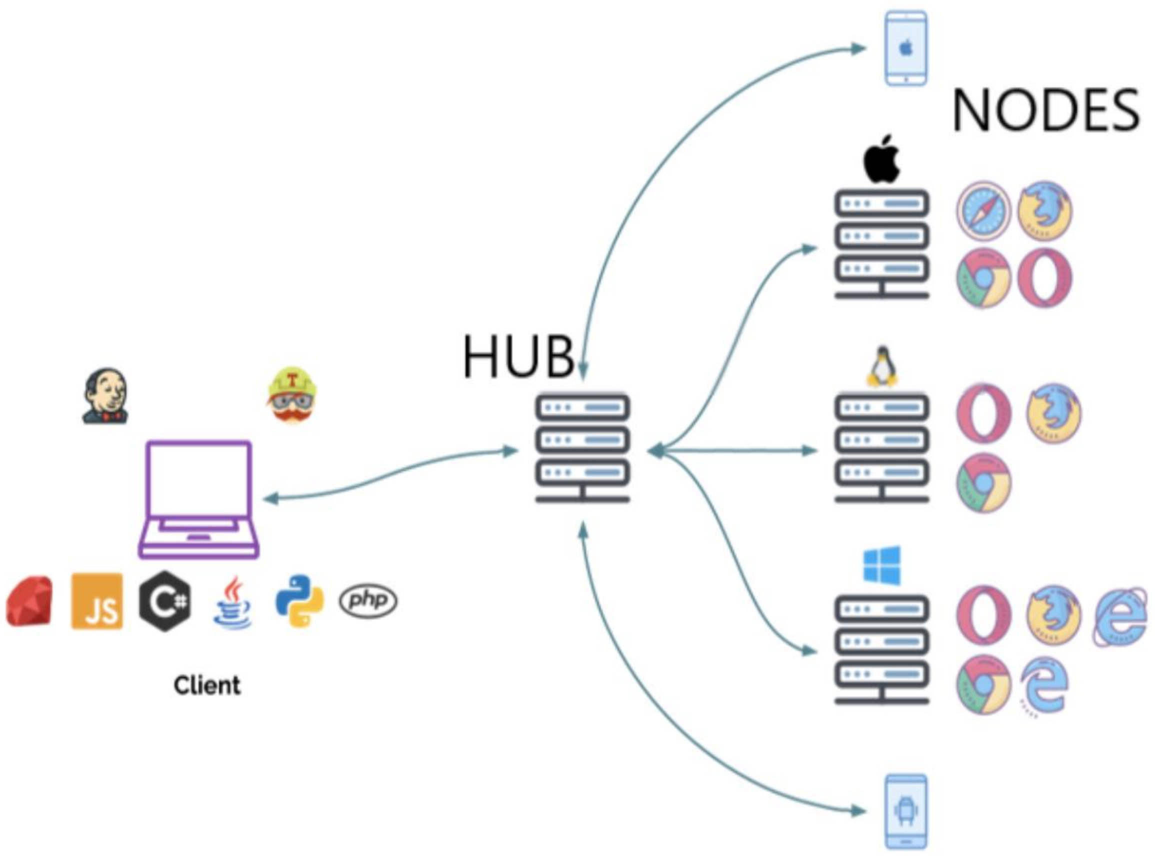
What is Selenium Grid?

Selenium Grid is a smart proxy server that makes it easy to run tests in parallel on multiple machines. This is done by routing commands to remote web browser instances, where one server acts as the hub. This hub routes test commands that are in JSON format to multiple registered Grid nodes.

Hub enables simultaneous execution of tests on multiple machines, managing different browsers centrally, instead of conducting different tests for each of them. Selenium Grid makes cross-browser testing easy as a single test can be carried on multiple machines and browsers, all together, making it easy to analyze and compare the results.

The two major components of the Selenium Grid architecture are:

- **Hub** is a server that accepts the access requests from the WebDriver client, routing the JSON test commands to the remote drives on nodes. It takes instructions from the client and executes them remotely on the various nodes in parallel
- **Node** is a remote device that consists of a native OS and a remote WebDriver. It receives requests from the hub in the form of JSON test commands and executes them using WebDriver



When should testers use Selenium Grid?
Testers should use Selenium Grid in the following circumstances:

- To run tests on multiple browsers and their versions, different devices, and operating systems
- To reduce the time that a test suite takes to complete execution

Selenium Grid improves the turnaround time of the test results. It is especially useful when the test suite is large and takes more time to run. It offers flexibility and ensures maximum test coverage within a limited time. Since the virtual infrastructure is in use, maintenance becomes easy.