Automation Testing

The “Automation Testing” automates the job of testing a software. In Automation Testing, a separate software is used to test the existing functional production software to be rolled out, based on the test cases identified. Automation Testing reduces the overall efforts and time required in regression testing and speeds up testing life cycle.

Automation Testing – WHY and WHEN?

- Frequent regression testing
- Virtually unlimited execution of test cases is required
- Rapid feedback to developers
- Reduction in human efforts
- Test same application in multiple environment
- Finding defects missed in manual testing

Automation Testing – Disadvantages

- High Initial Investment
- High Maintenance Cost
- Skill requirement
- Higher Timelines before use
- Long Payback Period
- Test Scripts Quality
- How to derive long term value

Selenium

Selenium is one of the most well-known testing frameworks in the world that is in use. It is an open source project that allows testers and developers alike to develop functional tests to drive the browser. A functional testing tool for web applications. It runs tests via a real browser that is driven by a JavaScript engine which is called "the BrowserBot”. Works with any JavaScript-enabled browser “, since Selenium has been built using JavaScript. It can be used to easily record and play tests.

Features of Selenium

- Allow Cross browser testing (Record in Firefox, Execute in IE)
- No dedicated machine required for test execution(user can work in parallel)
- Selenium uses JavaScript and IFrames to embed the BrowserBot in your browser

- The engine is tweaked to support wide range of browsers on Windows, Mac OS X and Linux

Selenium Core

Selenium Core is a JavaScript-based test tool for Web applications. Selenium Core tests run directly in a browser, just as real users do

- Utility for running tests in web browser
- Executes commands received from test script
- Allows test scripts to run inside supported browsers
- Works with Java script enabled browser
- Works on a large selection of browsers and operating systems
- Controls AUT (application under test) in other frame

Selenium RC

Selenium Remote Control (RC) is a test tool that allows you to write automated web application UI tests against HTTP website using any mainstream JavaScript-enabled browser

Selenium RC consists of two parts:

- Selenium Server: Works as an http proxy for web request
- Client Libraries: Client library for selected language for automation

Web Driver

WebDriver is an API designed to provide a simpler, more concise programming interface in addition to addressing some limitations in the Selenium-RC API. Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded. WebDriver's goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.

Selenium IDE

Selenium IDE (Integrated Development Environment) to develop automation scripts using selenium. Firefox extension. Record and playback test in browser. Intelligent field identification with IDs, names, XPaths etc. Record and walk through the test modes. Import and export scripts in multiple formats e.g. HTML, Ruby, Java, C#, Perl and Python. Allows script editing.

Selenium Grid

Selenium Grid is basically a tool used along with Selenium RC to run test suits in multiple environments and to run them parallel.

Features of Selenium Grid

- It enables concurrent running of test suits in multiple browsers and environments

- It's a time effective technique of running tests
- It works on the basis of hub and nodes concepts

WebDriver

"Web Driver" is a Web Automation Framework which is also known as "Selenium 2". It allows you to create and execute tests against different browsers, unlike Selenium IDE which works only with Firefox

WebDriver is designed to provide a simpler, more concise programming interface in addition to addressing some limitations in the Selenium-RC API. Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded.

WebDriver's goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.

Initialization of WebDriver

```
IWebDriver chromeDriver = new ChromeDriver();
```

To open an URL using any of the WebDrivers, use GoToUrl() method of INavigation interface

```
chromeDriver.Navigate().GoToUrl("http://google.co.in");
```

Limitation of WebDriver

Web Driver cannot support new web browsers out of the box

- Web Driver controls browser from OS level
- Different web browsers communicate with the OS in a different way
- New browsers may have different way of communicating with OS in a different way

No built-in test result generator support

- Selenium RC automatically generates the test result in an HTML format
- Web Driver has no built-in provision that can help tester in generating Test Results File
- The Tester would have to rely on your IDE's output window, or design the report yourself using the capabilities of your programming language and store it as text, html, etc

WebDriver Properties and Methods

Properties

Title – Gets the title of the Current browser window
Url – Get or sets the url the browser displays currently
PageSource Gets the source of the page last loaded by the browser

Methods

Close() – Closes only the current window
Quit() – Closes all windows opened by the IWebDriver
Manage() – Instructs the Driver to change its settings

Web Elements

WebElement represents an HTML element. All interactions with a page are performed through IWebElement Interface. FindElement command of IWebDriver returns IWebElement.

For e.g.

```
IWebElement element = driver.FindElement(By.Id("UserName"));
```

The above code snippet finds a web element with the Id UserName

IWebElement can be of any type, like it can be a Text, Link, Radio Button, Drop Down, WebTable or any HTML element. All the actions that populate against any of the element are common.

Locating UI Elements

By.Id - Locates element using value of their "ID" attribute
By.ClassName - Locates element using value of their "Class" attribute
By.Name - Locates element using value of their "Name" attribute
By.LinkText - Finds a link element by the exact text it displays
By.PartialLinkText - Find the link element with partial matching visible text.
By.CSS - Finds elements based on the driver's underlying CSS Selector engine
By.TagName - locates elements by their tag name
By. XPath - locates elements via Xpath

Forms using WebDriver

InputBox - SendKeys(), Clear()

RadioButton, CheckBox – Click()

Links – Click()

Drop-Down Box – Select()

Submit Form – Submit()

Handle Dynamic WebTables

In most cases, tables contain text data and you might simply like to extract the data given in each row or column of the table. But sometimes tables have link or images as well, and you can easily perform any action on those elements if you can find the HTML location of the containing cell.

All you need to is to inspect the table cell and get the HTML location of it using Xpath. To find and locate the table using XPath.

IWebElement myTable = driver.FindElement(By.XPath("//*[@id='example-datatable']/tbody"));

To get all the rows from the table

IList<IWebElement> rows = new List<IWebElement>(myTable.FindElements(By.TagName("tr")));

Locators

Locators are the lifeblood of the tests. Using the right locator ensures the tests are faster, more reliable or has lower maintenance over releases. Locators are used to find and match the elements of your page that it needs to interact with. Selenium provides us different types of locators.

- ID
- Name
- Class Name
- CSS Selector
- XPath
- Link Text
- Partial Link Text
- Tag Name

XPath

Language that describes a way to locate and process items in Extensible Markup Language (**XML**) documents by using an addressing syntax based on a path through the document's **logical structure or hierarchy**. XPath is used in Selenium to uniquely identify an element on a Webpage as element locator.

Absolute XPath

It starts from the root element of the HTML page. The root element for every HTML Page is "html"
It starts with "/" (single slash).

/html/body/div[2]/div[1]/div/div/div[5]/input

Relative XPath

It starts with "//" (double slash).

//input[@id='UserName']

Wait Command

Wait is a very important concept in any automation project. Waits are very vital to any project primarily because of the unpredictability that comes with automation. In case of Web application this unpredictability increases quite considerably, because of the below factors.

- Slow internet connection
- Ajax calls made by elements
- Data base calls made by website
- Slow browsers
- Delay loaded elements, which are used to optimize bandwidth usage

Most of the new learners use Thread.Sleep(); comment to make web driver wait for some seconds but it's not a good practice for real time testing.

For this Selenium Webdriver comes with a some really good inbuilt waits. We have different wait types in selenium webdriver.

- ImplicitWait
- ExplicitWait
- SetPageLoadTimeout
- SetScriptTimeout
- Thread.Sleep(ms)

Implicit Wait

Implicit wait is used to inform WebDriver that there could be cases when some elements on the webpage will not be present instantaneously. In those cases you have to wait for some time before trying to find the element. Default wait time is 0; WebDriver will try to find the element only once and after that it will throw an exception if element is not found. Once we declare with our own time WebDriver will wait for those seconds till the element is found in page.

```
driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(20);
```

Explicit Wait

Wait is explicitly applied to wait for conditions. Effective kind of wait that has to be used for each element or specified element. Explicit wait tells the WebDriver to wait till expected conditions become true, if it's not satisfied it will wait for maximum timeout period before throwing a 'NoSuchElementException'.

Expected conditions is mandatory once we declare explicit wait.

```
WebDriverWait wait = new WebDriverWait(driver,TimeSpan.FromSeconds(30));
wait.Until(drv => drv.FindElement(By.Id("myCarousel")));
```

WebDriverWait is present in the *OpenQA.Selenium.Support.UI*namespace. This wait is a specialized form of DefaultWait class.

Switch Commands

Webdriver library contains a set of classes and methods that help us to communicate with the web browser. When you try to find an element and some popup window has appeared in between, you should switch to the required window to locate your Web Element, else the driver will be locating the element in the current window and finally, you'll get NoSuchElement Exception. To address the issue in the above scenario, we use Switch commands. Theres is a set of methods accessible in the webdriver API.

ActiveElement – Switches focus to element that has focus at the moment. If there's no element with focus, this method will switch to body element of the page.
Alert – Switches focus to currently active dialog box
DefaultContent – Switches focus to the first frame on the page or main document if page contains iframes.
Frame – Switches focus to frame by given index
ParentFrame – Switches focus to parent frame of currently selected frame
Window – Switches focus to the window with specified name

Windows Handler in WebDriver

A set of windows that webdriver can interact with is called windows handles.
Each window handle is identified by random string, which is different on each test run.
The different window handles supported in Selenium are:

- *alert*
- *frame*
- *Window*

Alert

Alert is a *popup window* that can be shown by running *javascript alert()*method.
Alerts are different from regular windows. The main difference is that alerts are blocking in nature
Types of Alert

- Simple Alert
- Confirmation Alert
- Prompt Alert

Selenium provides IAlert interface located in OpenQA.Selenium.IAlert to handle alerts
IAlert alert = driver.SwitchTo().Alert();

alert.Accept(); //It accepts the alert.
alert.Dismiss(); // It rejects the alert.
alert.Text; //It gets text from the alert box.
alert.Sendkeys("Text"); // This is used to send any text to alert box if its prompt alert.

Window

Multiple Windows are opened by clicking on a link which opens the page in a new browser window.

Selenium web driver keeps a track of how many windows it opened during a session. Selenium will not keep a track of the number of Windows that are opened manually.

When we have multiple windows in test automation, all we need to do is switching the focus from one window to another. Each window in Selenium WebDriver has a unique handle identifier; this allows you to differentiate windows.

To shift focus from Main Window to any sub window use the following command on WebDriver
driver.SwitchTo().Window(String WindowHandle);

Once the Switch command executed all the driver commands will go to the newly focused window. CurrentWindowHandle Gets the current window handle, which is an opaque handle to this window that uniquely identifies it within this driver instance.

String mainWindow = driver.CurrentWindowHandle;

WindowHandles Gets the window handles of open browser windows.
List<string> subWindow = driver.WindowHandles.ToList();

BDD

BDD is a software development technique that defines the user behavior prior to writing test automation scripts or the functional pieces of code. Behavior-driven development should be focused on the business behaviors your code is implementing: **the "why" behind the code.** It supports a team-centric (especially cross-functional) workflow. Behavior Driven development is mostly about technical insight and business knowledge. Behavior of the user is defined by a product owner/business analyst/QA in simple English. These are then converted to automated scripts to run against functional code.

Features of BDD

The use of BDD requires no particular tools or programming languages, and is primarily a conceptual approach. Behavior Driven Development (BDD) is a methodology for developing software through continuous example-based communication between developers, QAs and BAs.

The primary purpose of BDD methodology is to improve communication amongst the stakeholders of the project so that each feature is correctly understood by all members of the team before development process starts. It involves getting stakeholders and delivery team with different perspectives onto the same page and ensuring that all have the same expectations. BDD helps to focus on the user's needs and the system's expected behavior rather than focusing too much on testing the implementation.

BDD vs Traditional Automation

BDD	Traditional Automation
Its plain text and easy to understand	Its full of code and hard to understand
Its easy to understand by BA/QA/DEV/Automation Test Engineer and all will be in same page	The code are understood only by Automation test engineer (Some times Dev)
Since its plain text format, BDD can be shared even to stakeholders	Impossible
Easy to learn and implement	More knowledge is required while designing

BDD vs TDD

Behavior Driven testing is an extension of TDD. Like in TDD in BDD also we write tests first and the add application code. The major difference that we get to see here are

- *Tests are written in plain descriptive English type grammar*
- *Tests are explained as behavior of application and are more user focused*
- *Using examples to clarify requirements*

BDD is in a more readable format by every stake holder since it is in English, unlike TDD test cases written in programming languages such as Ruby, Java etc. BDD explains the behavior of an application for the end user while TDD focuses on how functionality is implemented.

BDD enables all the stakeholders to be on the same page with requirements which makes acceptance easy, as opposed to TDD. Changes on functionality can be accommodated with less impact in BDD as opposed to TDD.

BDD Implementation

The focus on behavior during development makes the test useful as verification that you're building the right feature. The phrasing is in business language, not in the system's internal implementation language. The goal of BDD is a business readable and domain-specific language that allows you to describe a system's behavior without explaining how that behavior is implemented.

Tests are written in the form of plain text features descriptions with scenarios typically written before anything else and verified by the non-technical stakeholders. No development skills are required for creating tests as tests are written in the ubiquitous language. Business Analysts can actively participate in the automated test cases review process and give their feedback to enhance them.

At the heart of BDD is a changing approach to unit testing and acceptance testing. Acceptance tests should be written using the standard agile framework of a User story:

"As a [role] I want [feature] so that [benefit]". Acceptance criteria should be written in terms of scenarios and implemented as classes:
Given [initial context],
when [event occurs],
then [ensure some outcomes].

BDD Tools

Cucumber

It is a Java framework for BDD, by its support for the particular set of interactions between team members and stakeholders. It lets us define application behavior in plain meaningful English text using a simple grammar defined by a language called ***Gherkin***.

Cucumber itself is written in ***Ruby***, but it can be used to "test" code written in *Ruby* or other languages including but not limited to *Java*, *C#* and *Python.* Cucumber supports writing specifications in about 30 spoken languages, making it easy to deliver better for teams outside of English-speaking territories or those working on internationally targeted software.

SpecFlow

SpecFlow is inspired by *Cucumber* framework in the Ruby on Rails world. *Cucumber* uses plain English in the Gherkin format to express user stories. Once the user stories and their expectations are written, the Cucumber gem is used to execute those stores.

SpecFlow brings the same concept to the .NET world and allows the developer to express the feature in plain English language. It also allows to write specification in human readable ***Gherkin format***.

SpecFlow is a testing framework which supports Behavior Driven Development (BDD). SpecFlow is an open-source project. The source code is hosted on GitHub. It lets us define application behavior in plain meaningful English text using a simple grammar defined by a language called Gherkin.

Gherkhin

Gherkin is a language, that is used to write **Features, Background, Scenarios, and Steps**. Gherkin helps us to write concrete requirements. Gherkin files are plain text Files and have the extension '.feature'. Gherkin syntax is simple and readable. Each line that is not blank has to start with a Gherkin keyword, followed by any text you like.

The keywords are −

- Feature
- Background
- Scenario

- Given, When, Then, And, But (Steps)
- @ (Tags)

Feature

Feature acts as a heading for a Scenario. Feature has a list of steps to be performed scenario as a whole. The **Feature** keyword is used to describe a software feature, and to group the related scenarios. Feature files are the base files to hold all the scenarios

A Feature has three basic elements

- The keyword – Feature.
- The name of the feature, provided on the same line as the Feature keyword.
- An optional description that can span multiple lines

Background

Backgrounds are used with the scenarios which have common functionalities. They will be executed before every scenario and perform the operation specified.

For Example:

In scenarios like checking user settings, profile settings, user information, the common operations are

Launching the application

Login into the application

Scenario

Scenarios are just the declaration of operation in plain English Text. Scenario has many steps. Scenario holds Given-When-Then-And-But syntax. In order for Specflow to execute the test, we need to define the steps in separate file, here its .cs (C# Class) file.

Feature File

The main feature of the Specflow is that it focuses on Acceptance testing.

It made it easy for anyone in the team to read and write test and with this feature it brings business users in to the test process, helping teams to explore and understand requirements.

Feature: Sign up

Sign up should be quick and friendly.

Scenario: Successful sign up

New users should get a confirmation email and be greeted

personally, by the site once signed in.

Given I have chosen to sign up

When I sign up with valid details

Then I should receive a confirmation email

And I should see a personalized greeting message

Styles of Mapping

Mapping can be done in 3 styles in Specflow.

1. Regular Expression in attributes
2. Method Name-Underscore
3. Method Name-Pascal

Tags

Tags are markers that can be assigned to features and scenarios. Assigning a tag to a feature is equivalent to assigning the tag to all scenarios in the feature file. A Tag Name with a leading @ denotes tag. We can write our own definitions on what's going to happen when we tag a scenario (or feature) with a certain tag. If supported by the unit test framework, SpecFlow generates categories from the tags. The generated category name is the same as the tag's name, but without the leading @. You can filter and group the tests to be executed using these unit test categories.