Types of Applications

Desktop Application

- Application that runs stand-alone in a desktop system or a laptop
- May or may not require the web browser to run
- Example: Powerpoint, Media Player

Mobile Application

- Application designed to run on a mobile device
- May or may not require web browser to run
- Example: Music Player, Google Pay

Web Application

- Application that runs over a remote server
- Requires the web browser to run
- Example: Gmail, Facebook

Hybrid Application

- Application installed on a mobile device
- Internally a web browser is used to work with the application
- Example: Facebook App, Gmail App

What is Testing?

- Identifying bugs and missing components with respect to the actual requirements
- Verifying whether an application works as expected or not

Software Testing Methods

Manual Testing

Manual Testing

- Process of identifying bugs or defects in a software program
- Testing done manually without using any of the automation tools

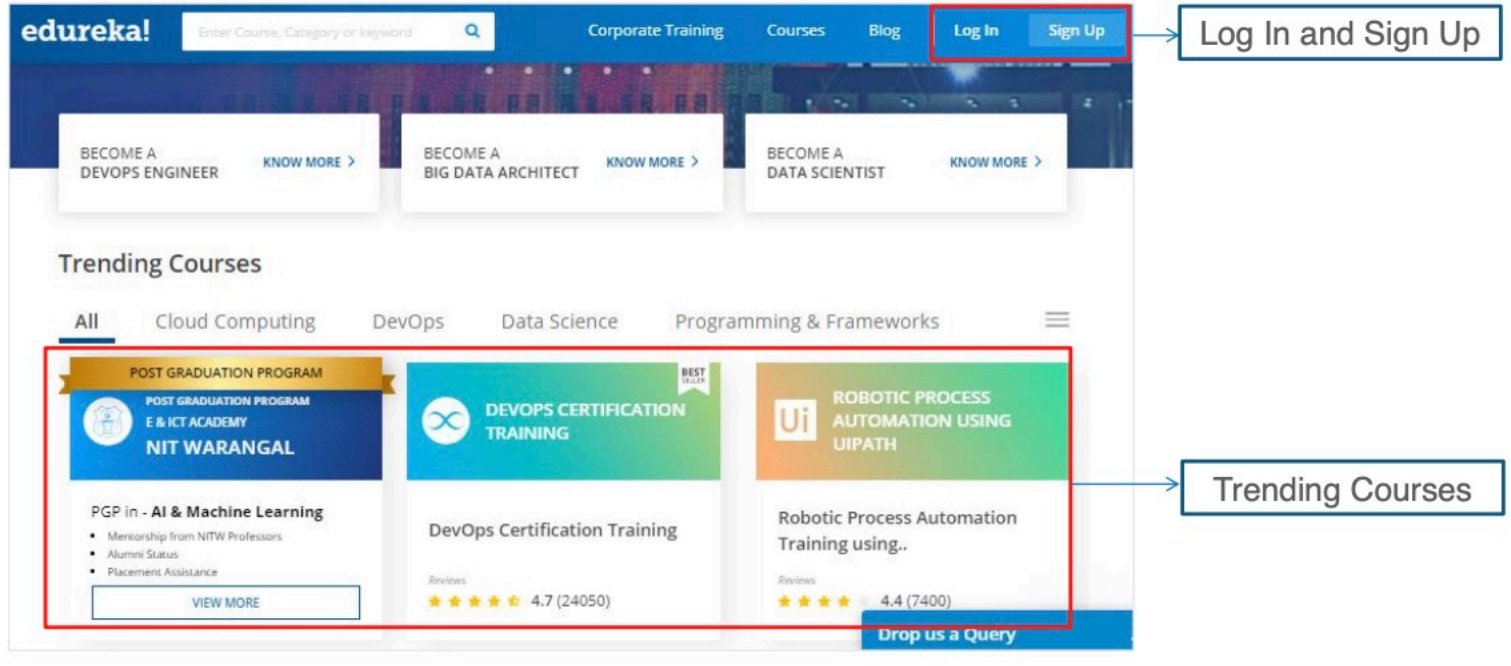
Test Automation

Test Automation

- Process of comparing actual test results with expected test results
- Testing done with the help of automation tools

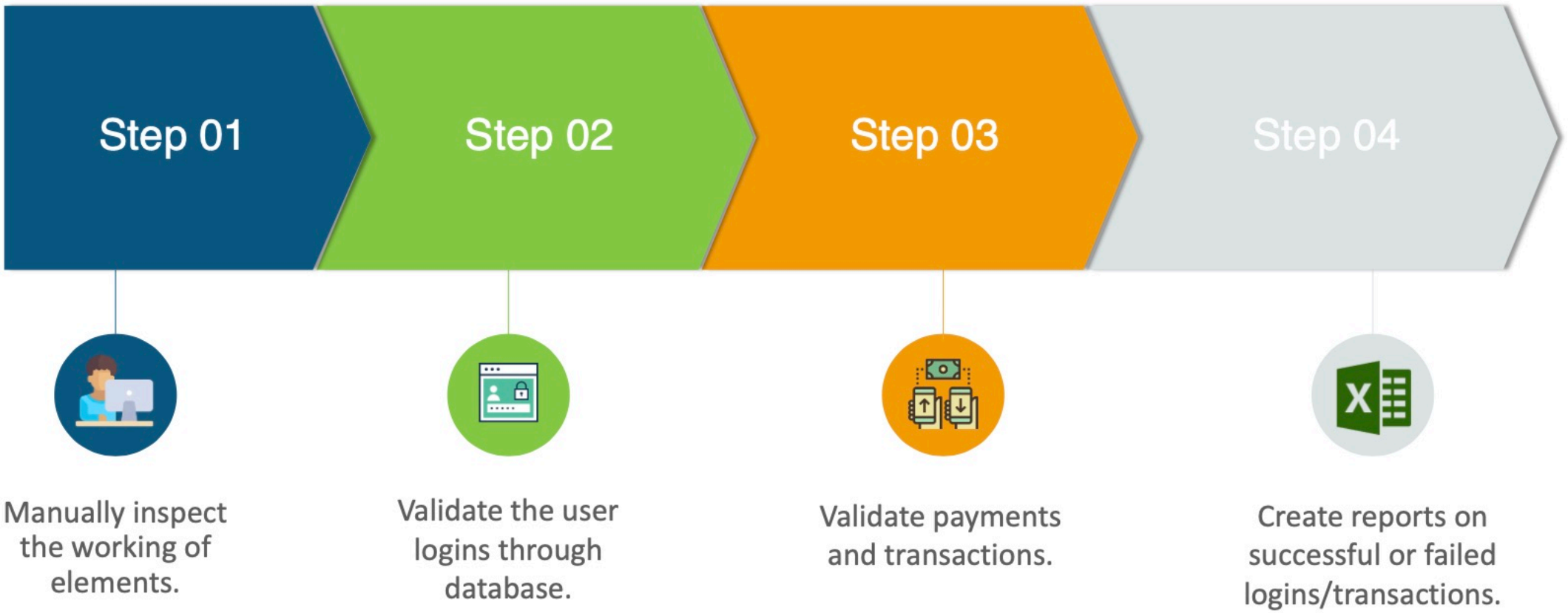
How Manual Testing Works?

- Consider an E-Learning Website wherein the elements of the website are to be tested manually
- Following are the elements of the E-Learning Website:
  - Log In and Sign Up
  - Trending Courses



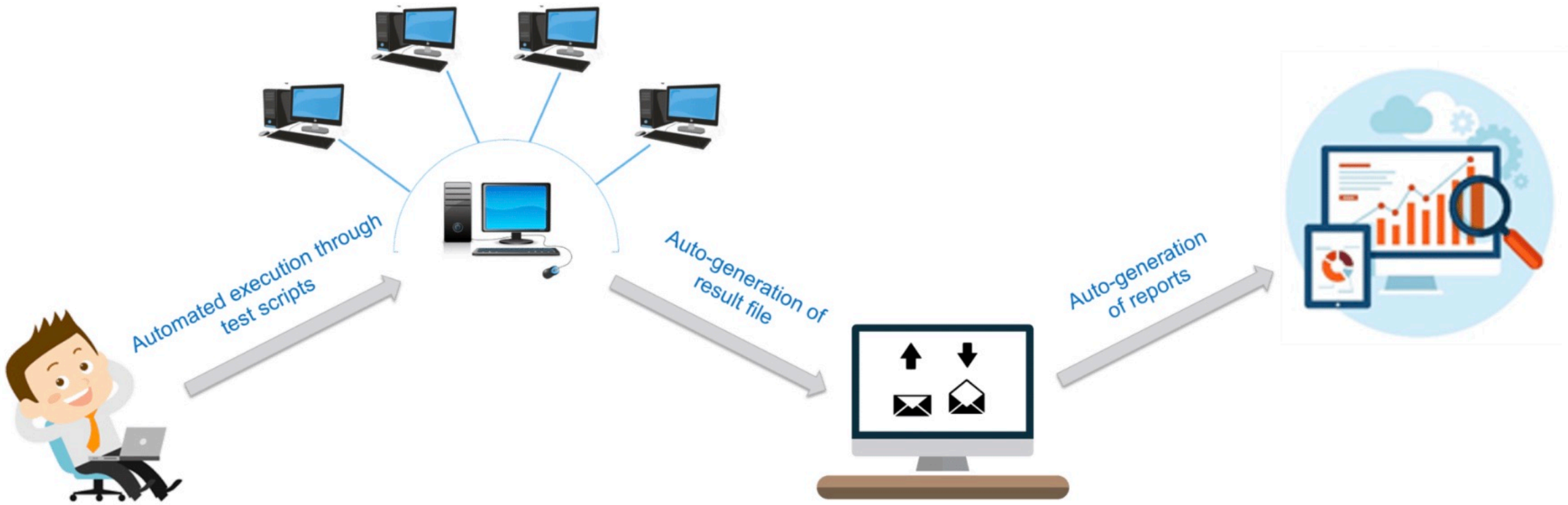
How Manual Testing Works? (Contd…)

Procedure followed in Manual Testing for an E-Learning Website:



Challenges in Manual Testing

How Test Automation Works?



Manual vs Test Automation

Properties	Manual Testing	Test Automation
Testing Time	Time Consuming	Significantly Faster
Human Resources	Huge Dependency	Less Dependency
Reliability	Less reliable since they are not accurate at all times	More reliable since they are performed using tools
Regression Testing	Not good for Regression Testing	Good for Regression Testing
Frequent Execution	Not a suitable option	Suitable option
UI Testing	Useful for UI Testing	Not useful for UI testing
Script Execution	Done Manually	Automated

Benefits of Test Automation

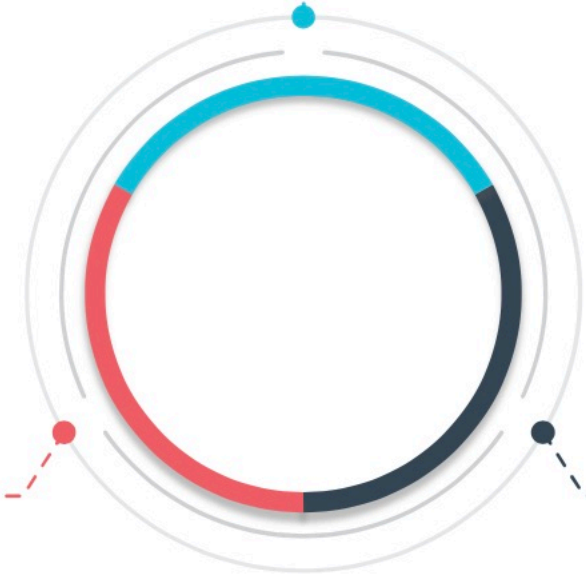
Types of Tests

Functional Tests	Non-Functional Tests
Functional Tests means testing the business logic behind an application Automation of Functional Tests means writing scripts to validate the business logic and the functionality expected from the application	Non-Functional Tests means testing the non-business requirements of an application Testing the requirements related to performance, security, databases and so on. These requirements remain constant or can be scaled as per the software size

Phases of Testing

Unit Testing

Units Tests are run during the development phase before handing over the system to the testers for testing



API Testing

API Tests are run during the Integration phase. Targets the Testing based on the request and response on which the application is built.

UI based Testing

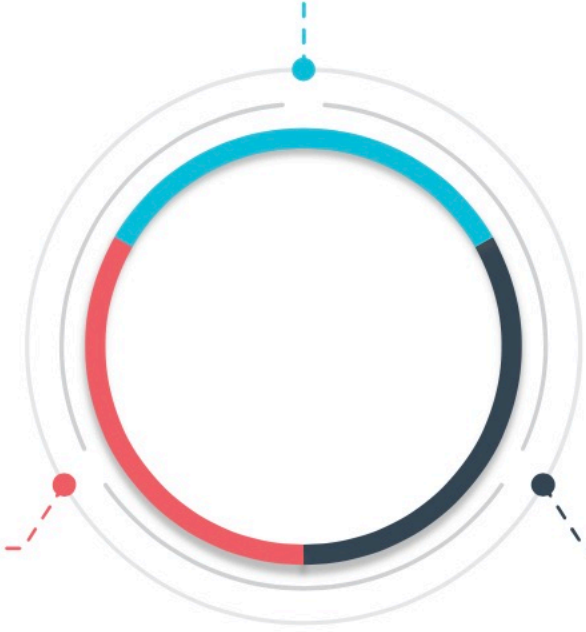
UI based Tests are run during the test execution phase.These test the business logic and functionality of the application from the front end

Test Automation Types

Automated Unit Testing

Testing at code level which involves identifying bugs in the functions, methods or routines

Example: JUnit tool integrated with Selenium



Automated API Testing

Testing the functionality, compliance and security issues

Example: SOAPUI tool

Automated GUI Testing

Testing of a User Interface of an application

Mostly used in Regression Testing scenarios

Example: Selenium, TestComplete tools

Test Automation Frameworks

- Test Automation Framework is a set of guidelines for creating and designing test cases
- Helps testers to use resources more efficiently

Linear Scripting (Record & Playback)

Manual recording is done at each step, checkpoints are inserted and the recorded script is played back.

Keyword-Driven Testing

Way of describing test cases using predefined set of keywords.

Modular Driven Testing

Divides the test scripts into small modules, wherein modules are small scripts written to perform certain tasks.

Hybrid Test Automation

Concept that uses the advantage of both Keyword and Data-Driven Framework.

Data-Driven Testing

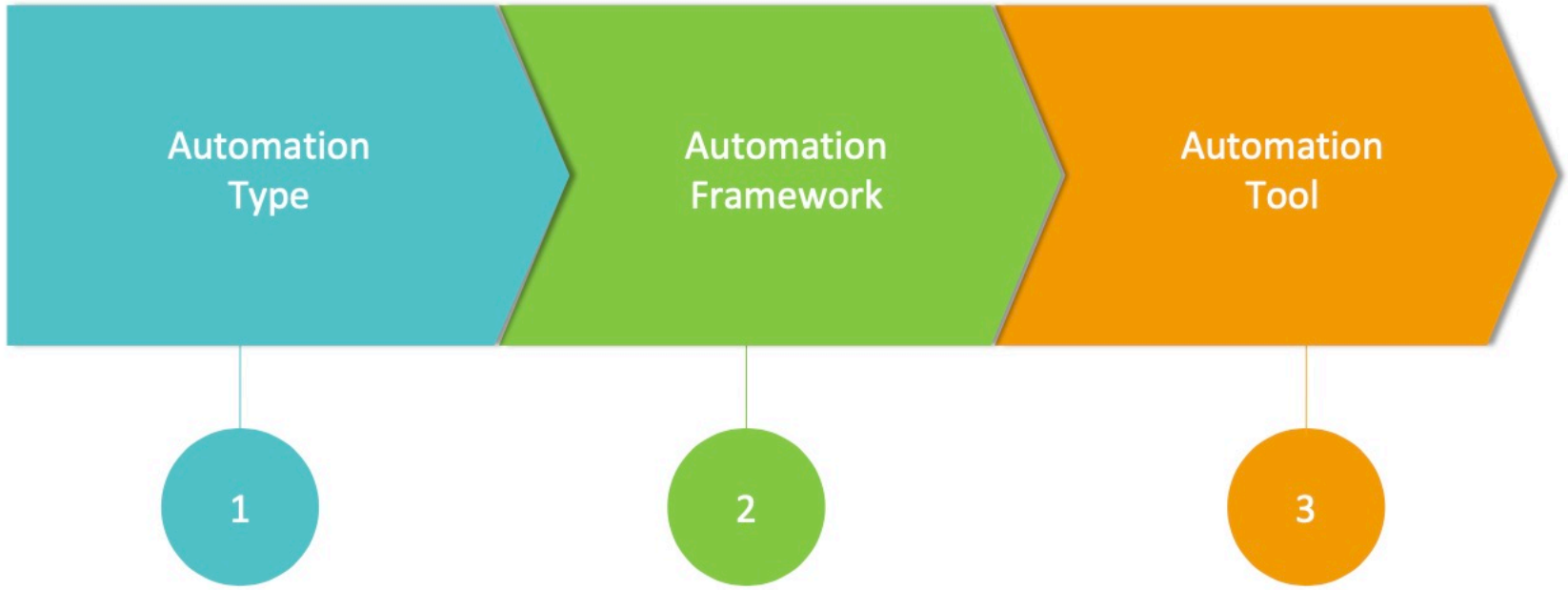
While Test Case logic resides in the Test Scripts, Test data is separated and kept outside the Test Scripts.

Behavior Driven Development Testing

Test cases are created in simple text Language (English).

Test Automation Tools

Test Automation Process



Identify the type of tests to automate.

Plan a design of automation and move towards a framework.

Select a tool based on the Automation Type and Automation Framework chosen.

What is Selenium?

Selenium is a tool which is used to automate web applications across different platforms using different programming languages

It is **open source** and mainly used for functional and regression testing

Known for better performance and execution speed

Best tool for testing web applications

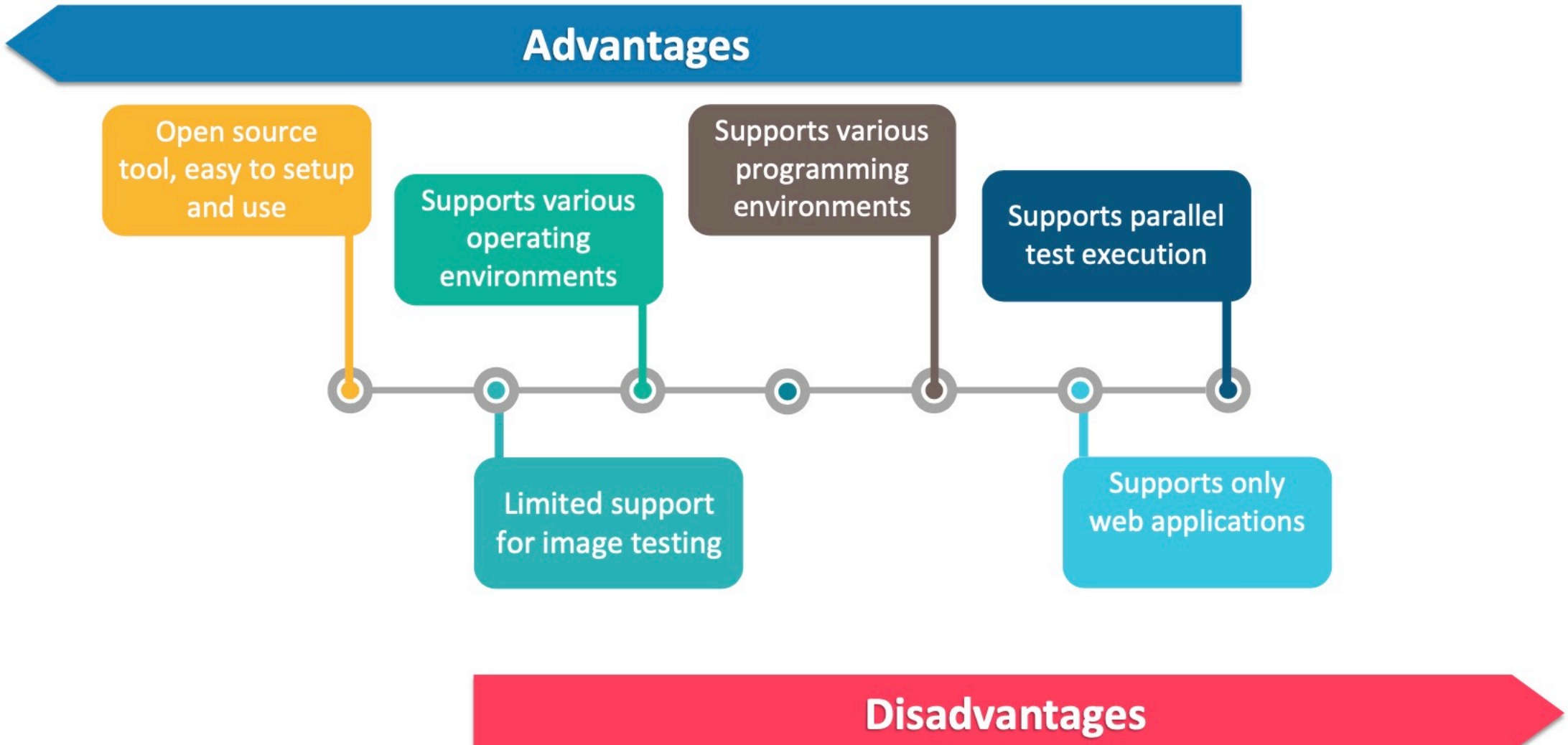
Selenium Supports the Following

Programming Languages

Operating Systems

Browsers

Advantages and Disadvantages of Selenium



Types of Testing Supported by Selenium

Selenium supports two types of Testing

Types of Testing: Functional Testing

Functional Testing

Regression Testing

Testing each and every function inside an application to ensure that it is working as per the requirement

Types of Testing: Regression Testing

Functional Testing

Regression Testing

Re-testing all the functionality when a new feature is added in the application to ensure all the functions work good with the new feature

Selenium vs Other Testing Tools

Features	QTP	IBM RFT	Selenium	TestComplete
License	Required	Required	Open Source	Required
Cost	High	High	Free	High
Customer Support	Dedicated HP support	Dedicated IBM support	Open Source Community	Support is provided
Hardware resource Consumption during Script execution	High	High	Low	High
Coding experience	Not Much	Required	Required along with technical capabilities of integrating the framework	Required
Environment support	Only for Windows	Only for Windows	Windows, Linux, Mac	Windows 7, Windows Vista, Windows Server 2008 or later OS
Language Support	VB Script	Java and C#	Java, C#, Ruby, Python, Perl, PHP, JavaScript	VB Script, JS Script, Delphi Script, C++ Script and C# Script

Components of Selenium Suite

Selenium IDE

Selenium WebDriver

Selenium Grid

- Selenium IDE (Integrated Development Environment) is mainly a Firefox and Chrome plugin
- It does simple record-and-playback of interactions with the browser
- Using IDE, program code can be exported in different languages: Java, Ruby, Python and so on

Few Drawbacks of Selenium IDE:

- Does not extend support to mobile based testing such as iPhone/ Android Testing
- Report generation is not as good as compared to other components

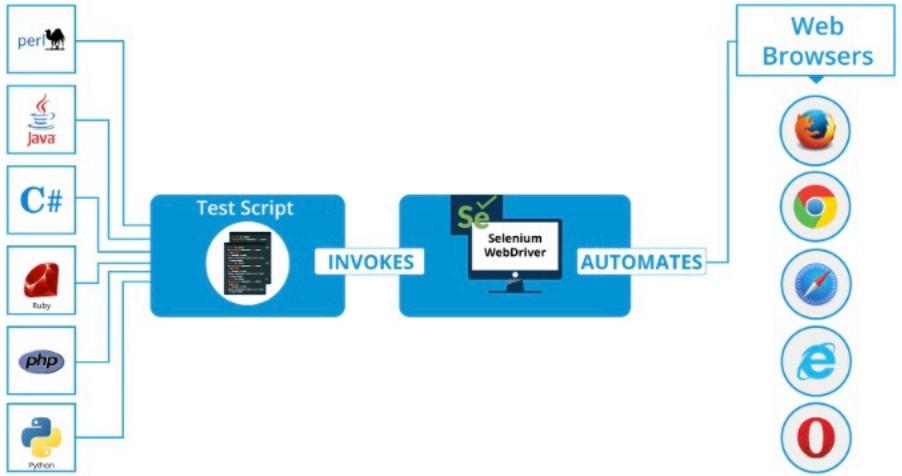
Components of Selenium Suite

Selenium IDE

Selenium WebDriver

Selenium Grid

- Selenium WebDriver is a browser automation framework that accepts commands and sends it to the browser
- Implemented through a browser-specific driver
- Directly communicates to the browser and controls it



Selenium WebDriver supports various operating systems as well such as Windows, Mac OS, Solaris and Linux

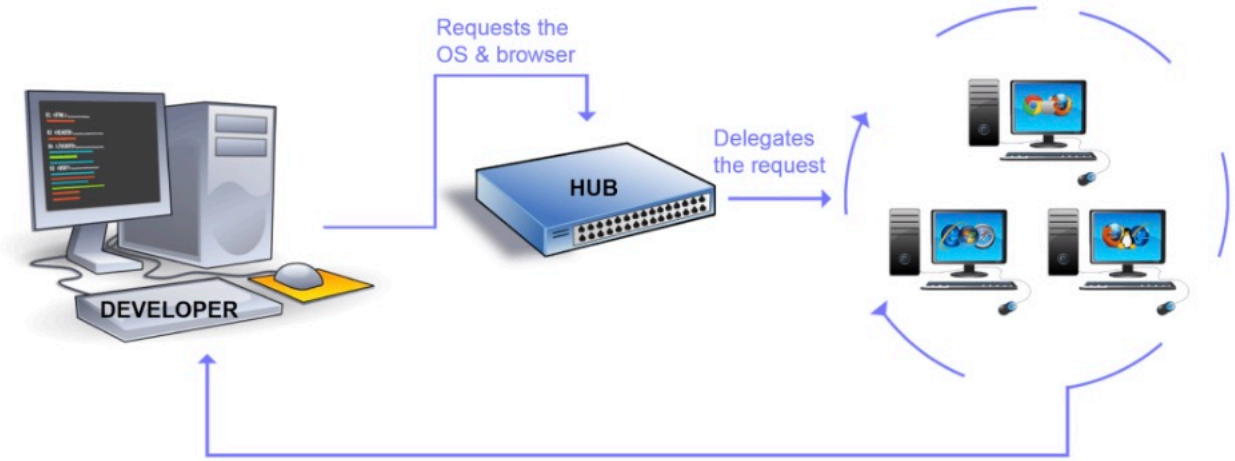
Components of Selenium Suite

Selenium IDE

Selenium WebDriver

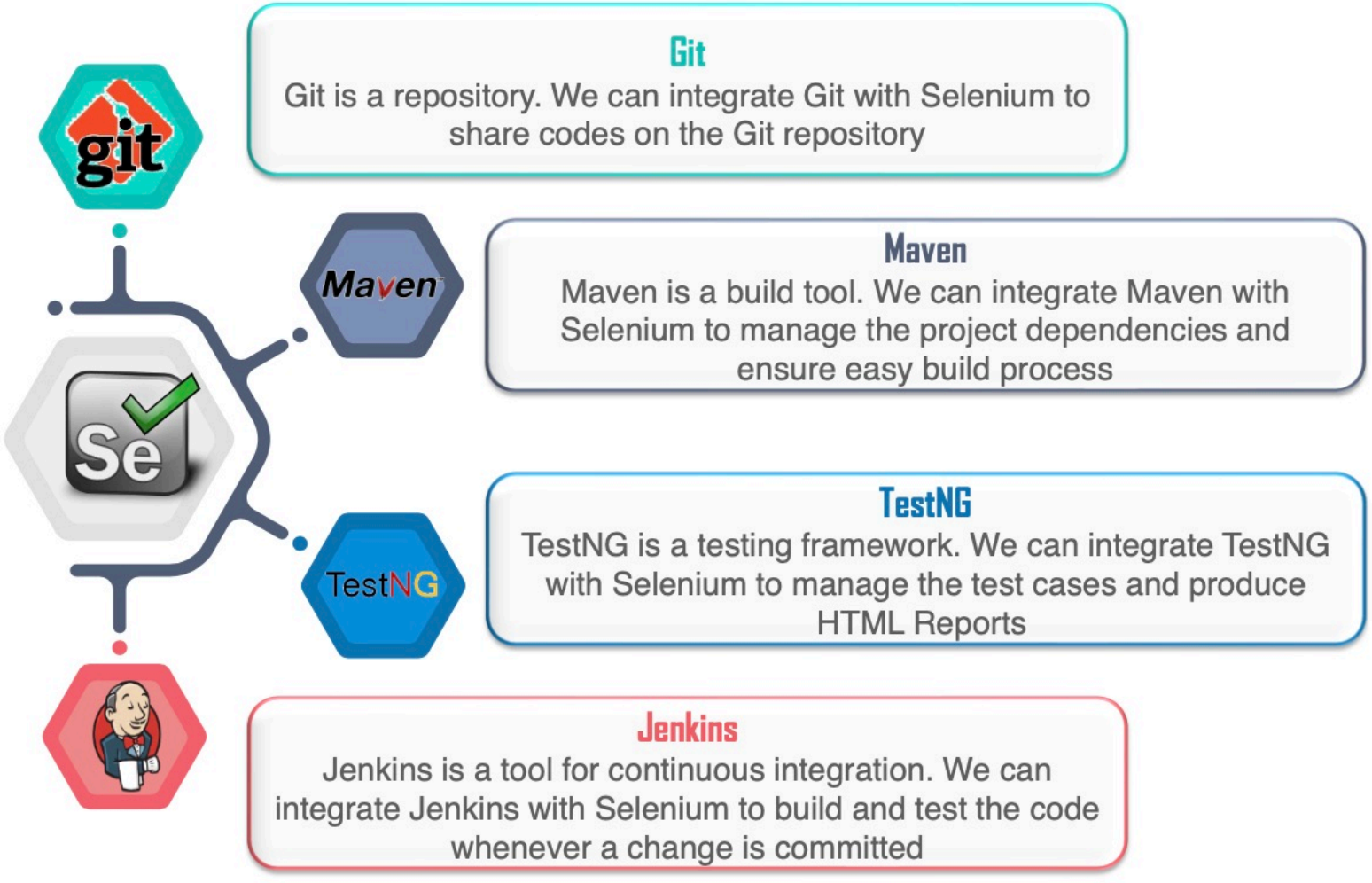
Selenium Grid

- Selenium Grid is used for parallel testing or distributed testing. It allows to execute test scripts parallelly on different machines.

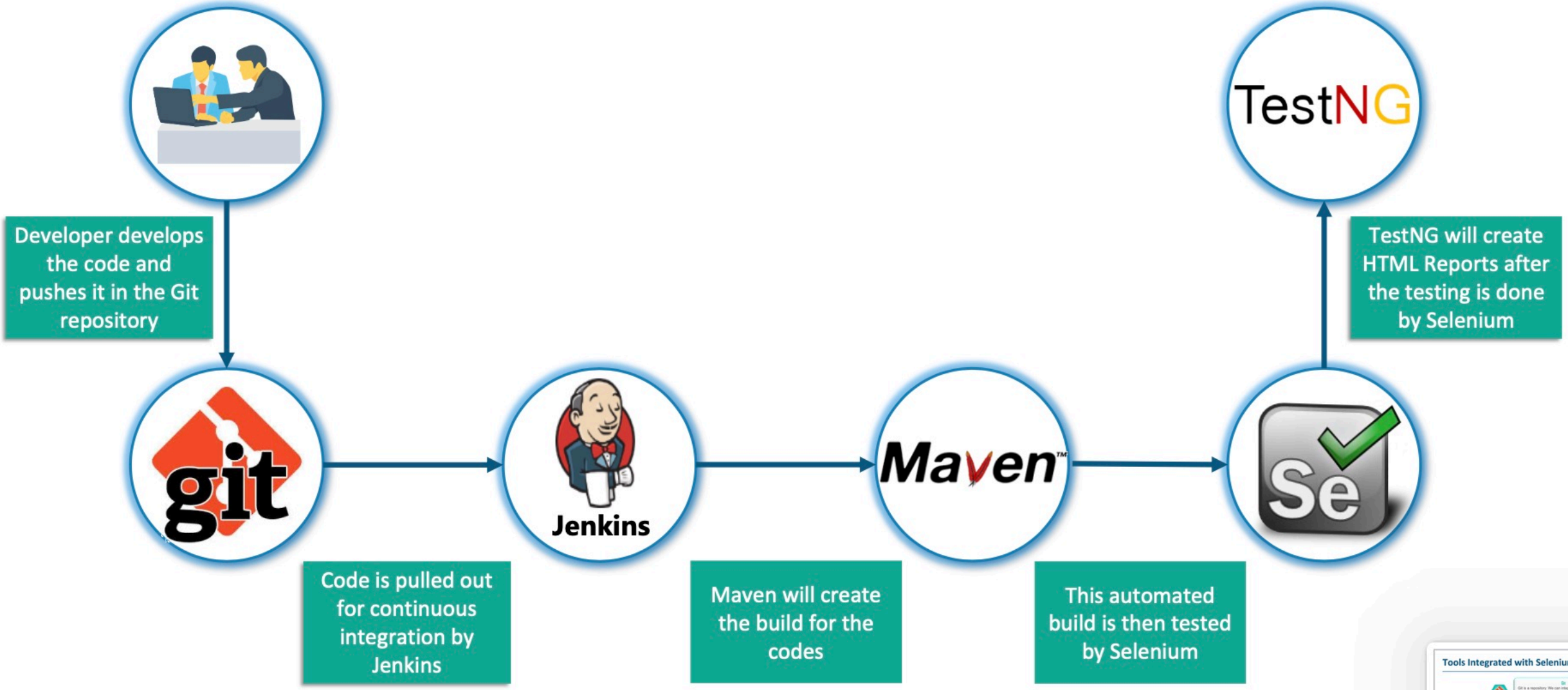


Parallel Execution → **Platform Independent** → **Language Independent** → **Browser Independent** → **Fast Execution**

Tools Integrated with Selenium



Integration of Selenium with Other Tools



Selenium WebDriver

Selenium WebDriver is a framework consisting of several open-source APIs that are used to automate tests on web-based applications

Advantages of Selenium WebDriver

Open Source

- Provides most of the features of a paid tool in a free of cost package
- Low maintenance cost

Language Support

- WebDriver supports all the major testing languages
- E.g.: Python, Java, C#, Ruby, PHP etc.

Cross-Platform

- Works across multiple platforms:
  OS: Windows, MAC, Linux
  Browsers: Chrome, Firefox, Edge, etc.
- Can test android and iOS web-apps

Frameworks & Plugins Support

- Supports various external frameworks such as Maven, ANT, TestNG, and Jenkins
- Customizable add-ons increase the scope of testing

Community Support

- Selenium is almost entirely community-driven project
- Updates are readily available

Advantages of Selenium WebDriver (Contd.)

Easy to Implement

- User-friendly implementation
- Allows scripting custom extensions to develop custom actions for the web-app

Mouse & Keyboard Simulation

- WebDriver enables imitating mouse and keyboard actions
- Advanced User Interactions API enables to perform such actions

Developer Code Leverage

- It can directly utilise the development code to speed up the test cycle
- Reduces redundancy of browsing through the application

Servers Not Required

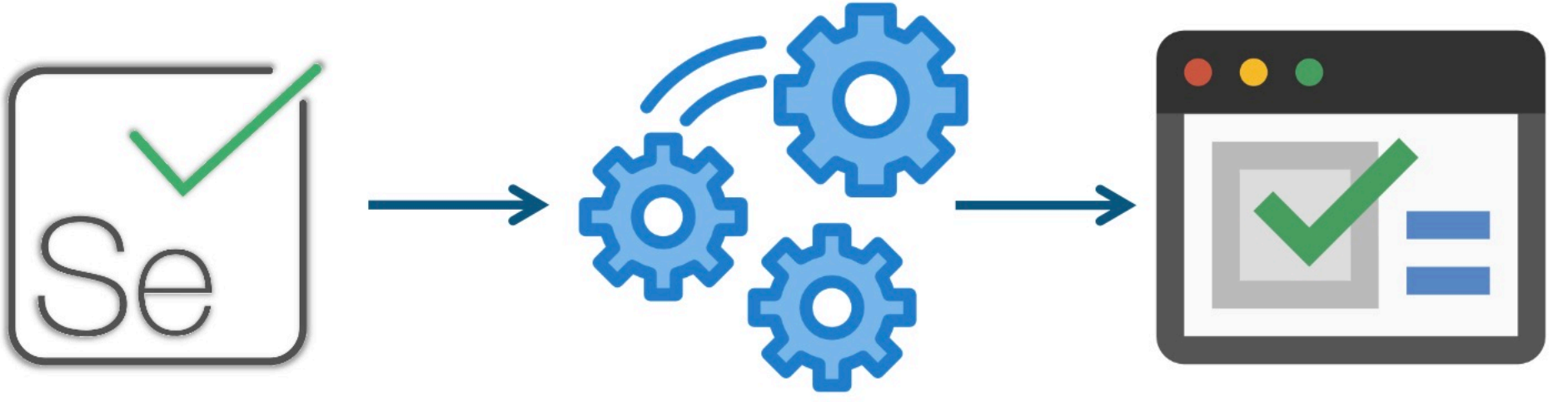
- WebDriver works directly through Webservices
- Requests are received by remote driver via HTTP and response is generated

Advanced Browser Interactions

- Interactions with Browser specific utilities is possible
- E.g. Useful in testing fintech applications

Selenium Browser Drivers

- Selenium requires a Web Driver Client to interact with the web application on the browser
- Driver is based on the browser where the application has to be tested



Selenium Browser Drivers

GeckoDriver (Mozilla Firefox)

Following the release of Mozilla Firefox v47.0.1, Mozilla introduced GeckoDriver to communicate with Selenium WebDriver 3.x and up

ChromeDriver (Google Chrome)

ChromeDriver by chromium is a standalone server that executes WebDriver on Desktop as well as Android version of Chrome web browser

Safari Driver (Apple Safari)

Safari Driver is the Selenium WebDriver interface for Safari Browser. It isolates the test execution from normal browsing data

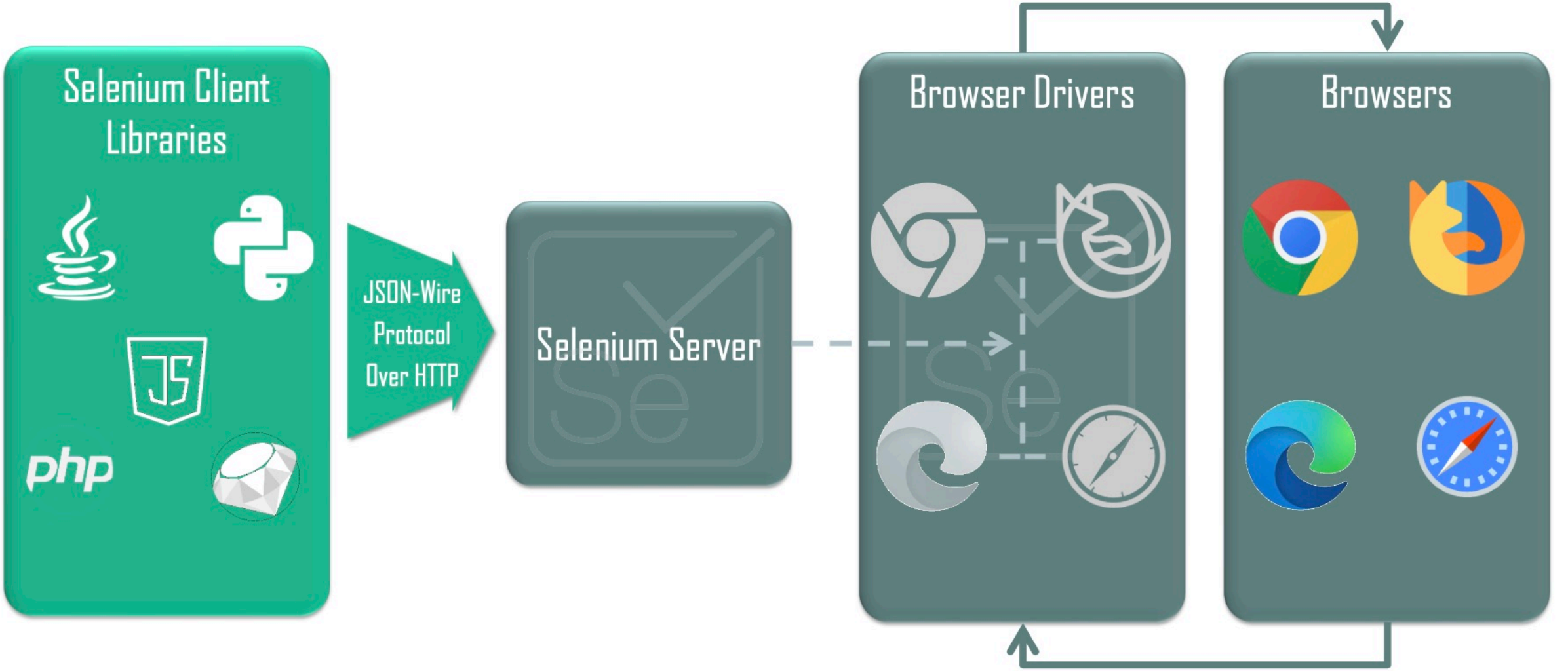
Microsoft Edge WebDriver (Microsoft Edge)

The WebDriver for Microsoft Edge Browser can be used to simulate User/OS level events on the browser

OperaChromiumDriver (Opera Browser)

The OperaChromiumDriver is a derived version of ChromeDriver to automate tests on the Opera Web Browser

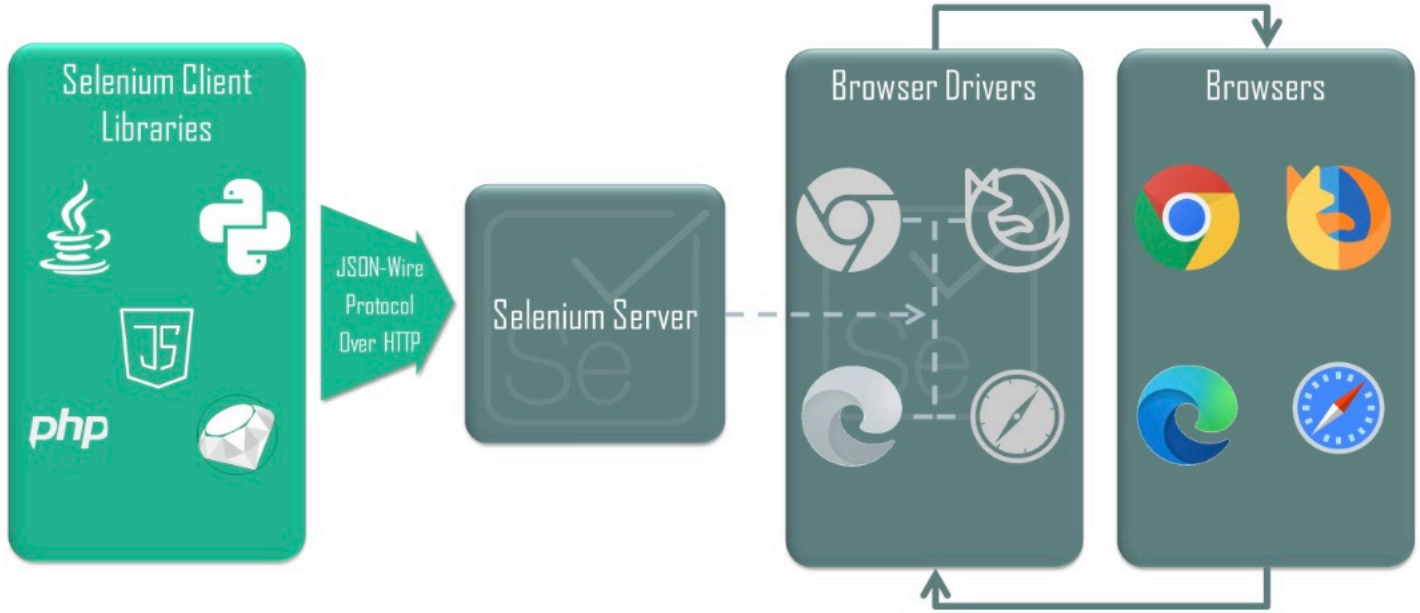
WebDriver Architecture



WebDriver Architecture

Selenium Architecture is divided into three layers:

- Client Libraries: Language Bindings
- WebDriver API
- Browser Drivers



WebDriver Architecture: Client Libraries

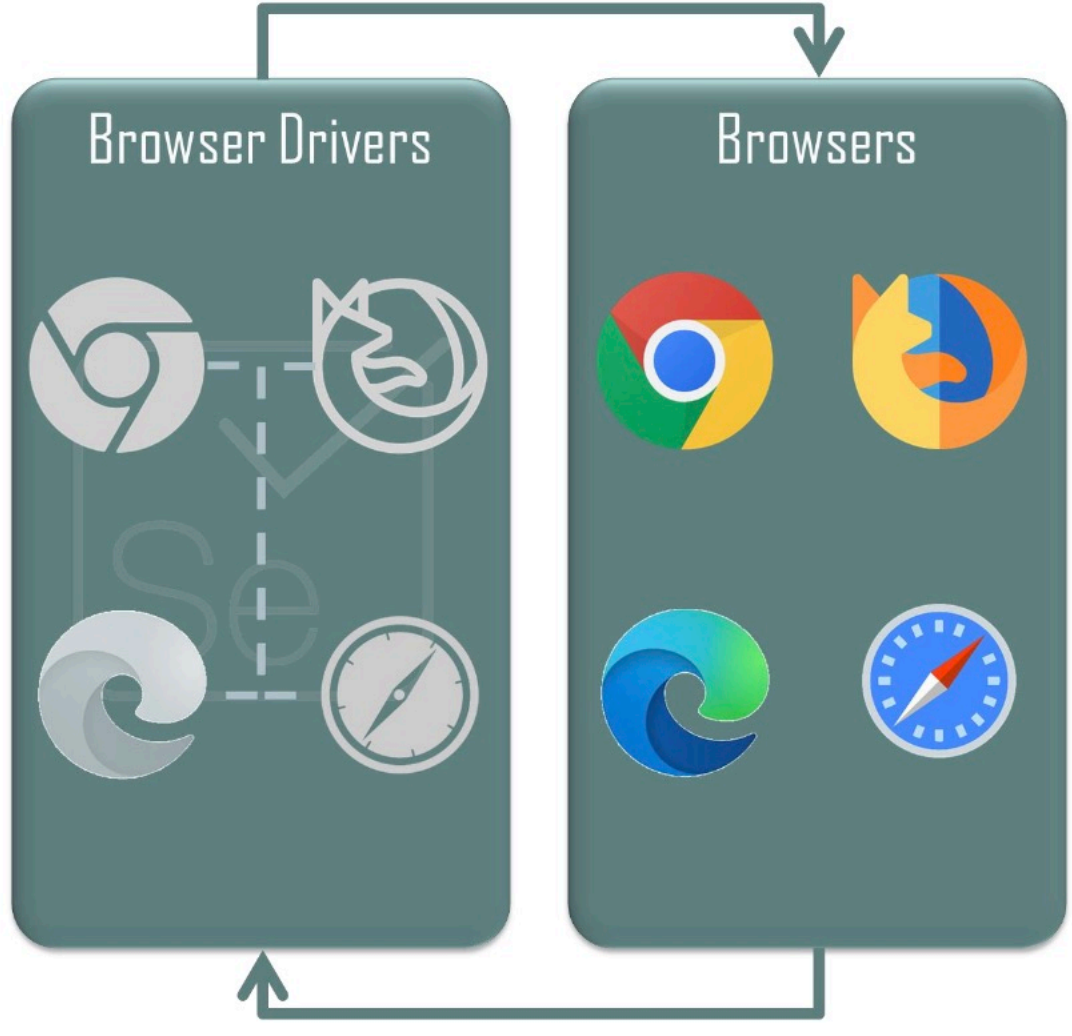
- Selenium Client Libraries enable support for multiple programming languages
- Language Bindings are used to develop frameworks which interacts with the Selenium WebDriver
- Language binding depends upon the browser driver used

WebDriver Architecture: WebDriver API

- JSON Wire Protocol is a REST API used to exchange information between HTTP servers
- It transfers information from the Client Libraries to the Browser WebDriver in JSON format
- WebDriver API interprets the language bindings
- The interpreted commands are sent to the browser drivers

WebDriver Architecture: Browser Drivers

- Browser drivers are made for specific browsers
- Drivers receive the commands from the server
- Drivers communicate with the browsers without revealing any internal browser logic
- After command execution, the browsers send an HTTP response



Web Elements

The GUI content of a web page is portrayed by Web Elements. A single web page could consist of many different web elements: TextBox, Labels, Buttons, Drop-Down menu, Hyperlinks, Check Boxes, etc.

Common Types of Web Elements

Text box
A graphical control element enabling user to input text

HyperLink
A highlighted text element which lets users navigate on click action

Button
Represents a clickable button usually attached to an action event

Check Box
GUI element which enables users to select multiple items from a list of options

Drop Down Menu
An interactable UI element which enables users to choose from a list

Radio Button
GUI element which enables users to select a single item from a list of options

Locating Elements using WebDriver

In order to automate tests on a web page, Selenium first needs to locate and identify the elements to interact with them

Locating Elements using WebDriver

Selenium WebDriver employs two methods to locate elements on a web page:

- FindElement
- FindElements

- Both the methods are used in conjunction with the By method
- FindElement(s) method instructs the WebDriver to start the search
- By method defines the element locating strategy

FindElement vs FindElements

FindElement Method	FindElements Method
Finds and returns the first element with the same locator	Searches and returns a list of all the elements matching the locator
Stops after locating the first element matching the locator	Keeps looking for more elements on the web page till the entire page is scanned
If no element is found; throws an NoSuchElementException	Returns an empty list in-case no element matching the locator is found

Syntax: FindElements & FindElements

FindElement Syntax

```
WebElement elementName = driver.findElement(By.LocatorStrategy("value"));
```

FindElements Syntax

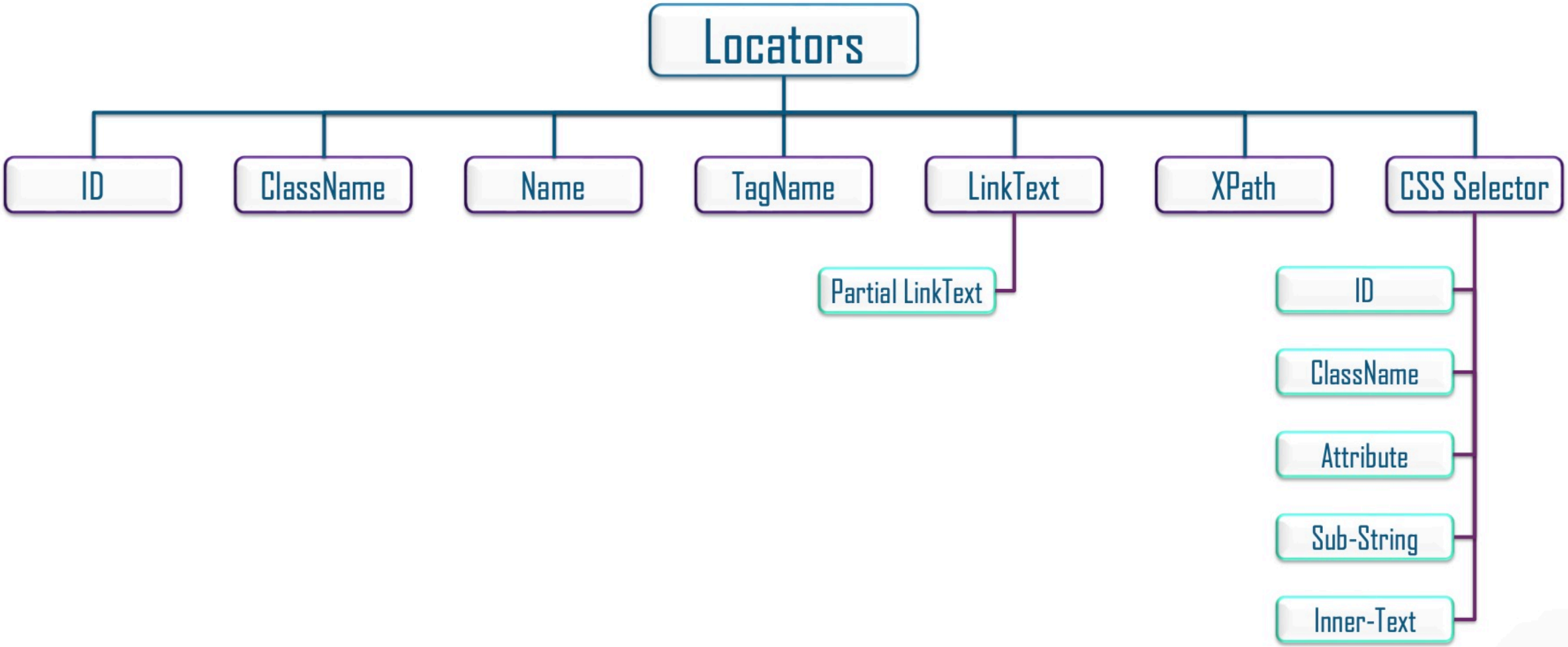
```
List elementName = driver.findElements(By.LocatorStrategy("value"));
```

Locators in Selenium

- Locators are addresses to distinctly identify web elements on a web page
- The address is usually an HTML property of the element
- Selenium uses locators to perform the required actions on the elements

```
Xpath=//tagname[@attribute='value']
```

Types of Locators



Types of Locators

- ID
- ClassName
- Name
- TagName
- LinkText
- XPath
- CSS Selector

- ID locators are unique to each element
- According to W3C standards, IDs should always be unique on a page
- The fastest, most reliable method to locate web elements on web page

```
Syntax: findElement(By.id("IdName"))
```

Note: W3C is the international standards organization for World Wide Web.

Types of Locators

- ID
- **ClassName**
- Name
- TagName
- LinkText
- XPath
- CSS Selector

- ClassName locator matches the class attribute of the element on the page
- If multiple values are specified for a single element; using any one of the values will work
- Class attribute is also effective in identifying elements using CSS files as classes

```
Syntax: findElement(By.className("ElementClass"))
```

Types of Locators

- ID
- ClassName
- **Name**
- TagName
- LinkText
- XPath
- CSS Selector

- Name attribute is similar to ID attribute in implementation
- Unlike ID, multiple elements can have the same name attribute
- In such cases, WebDriver picks up the first element it finds

```
Syntax: findElement(By.name("name"))
```

Types of Locators

- ID
- ClassName
- Name
- **TagName**
- LinkText
- XPath
- CSS Selector

- Elements can be directly identified using HTML DOM tags
- Useful while extracting information from the tag itself
- Can be used in conjunction with other attributes

```
Syntax: findElement(By.tagName("HTMLTagName"))
```

Types of Locators

- ID
- ClassName
- Name
- TagName
- **LinkText**
- XPath
- CSS Selector

- Used to locate hyperlinked elements
- If multiple elements have the same link, the first element is picked
- Part of the link text(string) can also be used to identify the hyperlinked elements

Link Text Syntax: `findElement(By.linkText("LinkText"))`

Partial Link Text Syntax:
`findElement(By.partialLinkText("PartialLinkText"))`

Types of Locators

- ID
- ClassName
- Name
- TagName
- LinkText
- **XPath**
- CSS Selector

- XPath locator enables searching elements on a web page using XML Path expression
- The elements can be selected using a combination of element's attributes
- Reliable but slower from performance perspective

```
Syntax: findElement(By.xpath("XPath"))
```

Note: The topic XPath is expanded further in this module.

Types of Locators

- ID
- ClassName
- Name
- TagName
- LinkText
- XPath
- **CSS Selector**

- CSS uses a combination of HTML tags and attributes to locate the element
- More advanced than other locators in selenium
- Can be used to access elements with no ID/name attributes

```
Syntax: findElement(By.cssSelector("cssValue"))
```

Note: The topic CSS Selector is expanded further in the next module.

XPath

- XML Path a language that enables querying XML documents
- It includes an expression for a path with some condition to locate the element

<xpath>

XPath Syntax

Following is the general syntax for xpath:

Reference to the tag used to identify the element

The value field of the selected attribute

```
Xpath Syntax => //tagname[@attribute='value']
```

Selects the current Node

Attribute of the element you want to locate

Types of XPath

There are two types of XPaths:

Absolute XPath

Relative XPath

Types of XPath

Absolute XPath	Relative XPath
The expression is created from the root node	Expression is relative and can be referenced from the middle of the DOM structure
Uses a single slash '/' and starts from the root of node's structure	Uses double slash '//' to indicate the root node
High risk of test failures if the path is changed during development	Low risk of failure as the search starts from the mentioned tagName position

XPath Functions

XPath for certain complicated elements is hard to define. XPath functions ease the process of locating such elements

XPath Functions

Contains

Starts-with

Text

- Useful when part of the attribute value changes dynamically
- contains() can locate element using partial text of the attribute

```
XPath: //tagName[contains(@attribute,'partialValue')]
```

XPath Functions

Contains

Starts-with

Text

- Helps locating elements with some static initial string value
- Partial value may dynamically change if some event such as page refresh, button press, etc is triggered

```
XPath: //tagName[starts-with(@attribute,'initialString')]
```

XPath Functions

Contains

Starts-with

Text

- Helps locate elements with a specified string present in the HTML tag
- Used when dynamic aspect of the element doesn't allow the use of contains() or starts-with() method
- Can be used in conjunction with other functions

```
XPath: //tagName[text()='searchText']
//tagName[contains(text(),'searchText')]
```

Advanced XPath Methods

AND OR

Index

Chained

- Logical expressions can be used to introduce conditions in XPaths
- AND ensures both the conditions should be true to locate the element
- OR on the other hand only requires one condition to be true

```
And: //tagName[ @attribute1='value' and @attribute2='value']
```

```
OR: //tagName[ @attribute1='value' or @attribute2='value']
```

Advanced XPath Methods

AND OR

Index

Chained

- Index helps identify elements from a list of multiple instances of an HTML tag
- The user can specify the index value with the tag name
- Useful in isolating elements present in tabular structure

```
XPath: //tagName[@attribute='value'][indexValue]
```

Advanced XPath Methods

AND OR

Index

Chained

- WebDriver enables chaining two or more XPath expressions together to locate element
- Double slashes '//' are used to chain the XPath expressions

XPath:

```
//tagName[@attribute='value']//tagname[@attribute='value']
```

XPath Axes

- Axes can be employed when the element in question has dynamic attributes
- It helps locating element by traversing through either of the parent, child or sibling tags

XPath Axes

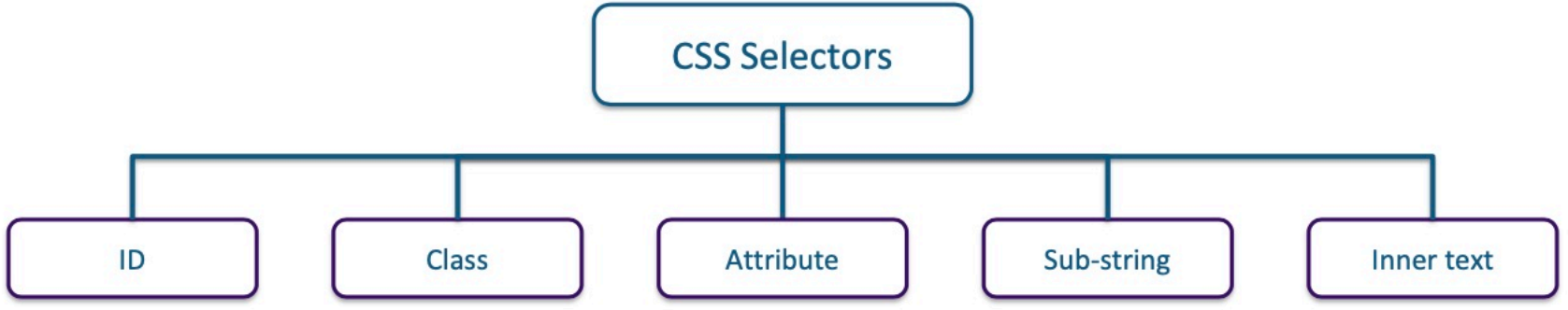
Function	Description
Child	Traverse all the child nodes of the context node
Parent	Selects the parent node of the context node
Following	Helps locate element which is following the current node
Following-sibling	Lets you navigate and locate an element from a sibling of a parent node
Preceding	Helps locating element which precedes the current node
Preceding-sibling	Navigates and identifies the parent sibling of the context node
Descendants	Identifies all the children as well as sub-children of the current node
Ancestors	Selects the ancestors i.e. parents/grand-parents of the context node

XPath Axes

Function	Syntax
Child	`//tagName[@attribute='value']//child::tag`
Parent	`//tagName[@attribute='value']//parent::tag`
Following	`//tagName[@attribute='value']//following::tag`
Following-sibling	`//tagName[@attribute='value']//following-sibling::tag`
Preceding	`//tagName[@attribute='value']//preceding::tag`
Preceding-sibling	`//tagName[@attribute='value']//preceding-sibling::tag`
Descendants	`//tagName[@attribute='value']//descendant::tag`
Ancestors	`//tagName[@attribute='value']//ancestor::tag`

What are CSS Selectors?

- CSS Selector is the combination of an element selector and a selector value which identifies the web element within the web page
- Also known as Selector Pattern



Locating Web Elements using CSS Selectors

- ID
- Class
- Attribute
- Sub-string
- Inner text

```
Syntax: css = <HTML Tag> <#> <Attribute Value>

Example:css = input#email

//Syntax to find the textbox via ID:

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

- <HTML Tag> is used to denote the web element
- # (hash) sign symbolizes the ID attribute
- Value of ID is preceded by a # (hash) sign

Locating Web Elements using CSS Selectors

ID

Class

Attribute

Sub-string

Inner text

```
Syntax: css = <HTML Tag> <.> <Attribute Value>

Example:css = button.radius

//Syntax to find the button via Class:

driver.findElement(BycssSelector("button.radius")).
click();
```

- <HTML Tag> is used to denote the web element
- . (dot) sign symbolizes the Class attribute
- Value of Class is preceded by a . (dot) sign

Locating Web Elements using CSS Selectors

- ID
- Class
- **Attribute**
- Sub-string
- Inner text

```
Syntax: css = <HTML Tag> <[Attribute = Attribute Value>

Example:css = input[name=email]

//Syntax to find the textbox via Attribute:

driver.findElement(BycssSelector("input[name=email]").
sendKeys("abc@gmail.com");
```

- Attribute for creating CSS Selector can be value, type, name and so on
- Select an attribute whose value uniquely identifies the web element

Locating Web Elements using CSS Selectors

- ID
- Class
- Attribute
- **Sub-string**
- Inner text

```
//Syntax to match a prefix:
     css = <HTML Tag> <[Attribute ^= prefix of the string]>
     css = input#email[name^='ema']

driver.findElement(BycssSelector(input#email[name^='ema']));

//Syntax to match a suffix:
     css = <HTML Tag> <[Attribute $= suffix of the string]>
     css = input#email[name$='ail']

driver.findElement(BycssSelector(input#email[name$='ail']));

//Syntax to match a substring:
     css = <HTML Tag> <[Attribute *= substring]>
     css = input#email[name*='ail']

driver.findElement(BycssSelector(input#email[name*='ail']));
```

- Match a prefix is used to correspond to the string with the help of a matching prefix
- Match a suffix is used to correspond to the string with the help of a matching suffix
- Match a substring is used to correspond to the string with the help of a matching substring

Locating Web Elements using CSS Selectors

ID

Class

Attribute

Sub-string

Inner text

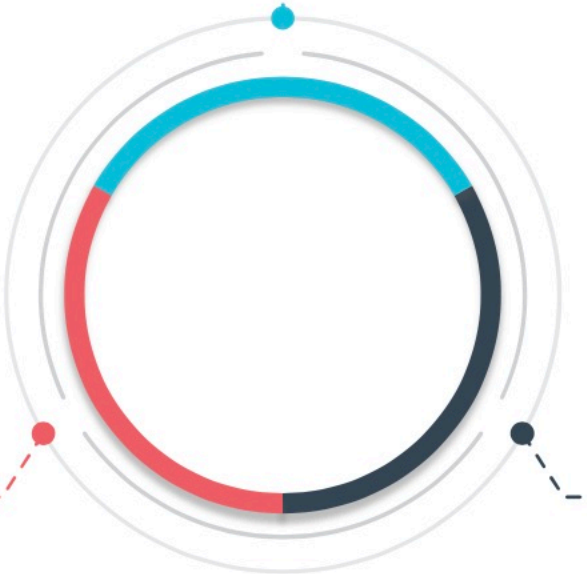
```
Syntax: css = <HTML Tag> <:><contains><(text)>

Example:css = font:contains("Password:")

//Syntax to find the label via Inner text:

driver.findElement(BycssSelector("font:contains("Passwo
rd:")"));
```

- : (colon) sign symbolizes the contains method
- contains is the value of a Class attribute which is being accessed
- text is the text that is displayed on the webpage at any location

Selenium Commands in WebDriver

Browser Commands

Basic browser operations of Web Driver such as opening a browser, performing few tasks and closing the browser



Navigation Commands

Basic browser navigation commands that allows the browser to move backward or forward in the browser history

Web Element Commands

Basic functions or methods for interacting with the Web Elements

Browser Commands in WebDriver

Command	Description
get(String arg0) : void	▪ Loads a new web page in the existing browser window
getTitle() : String	▪ Fetches the title of the current web page
getCurrentUrl() : String	▪ Fetches the string representing the Current Url of the current web page
getPageSource() : String	▪ Returns the source code of the current web page loaded on the current browser
close() : void	▪ Terminates the current browser window operated by WebDriver at the current time
quit() : void	▪ Terminates all windows operated by WebDriver

Navigation Commands in WebDriver

Command	Description
to(String arg0) : void	▪ Loads a new web page in the existing browser window
forward() : void	▪ Enables the web browser to click on the forward button in the existing browser window
back() : void	▪ Enables the web browser to click on the back button in the existing browser window
refresh() : void	▪ Reloads the current web page in the existing browser window

Web Element Commands in WebDriver

Command	Description
getAttribute()	▪ Fetches the attribute values of the attributes of a Web Element
getText()	▪ Returns the visible text if the element contains any text on it
getCSSValue()	▪ Fetches a CSS property value from a Web Element
getLocation()	▪ Returns the relative position of the rendered Web Element
getSize()	▪ Returns the width and height of the rendered Web Element
getTagName()	▪ Returns the HTML tag name of the Web Element

Methods to Interact with Web Elements

- **getAttribute()**
- getText()
- getCSSValue()
- getLocation()
- getSize()
- getTagName()

- getAttribute() method can be executed on all the Web Elements
- This method is used to fetch the attribute values of the attributes of a Web Element

API Syntax:

```
java.lang.String getAttribute(java.lang.String name)
```

- The input parameter is String, which is the name of the attribute
- The return type is String, which is the value of the attribute

Methods to Interact with Web Elements

- getAttribute()
- **getText()**
- getCSSValue()
- getLocation()
- getSize()
- getTagName()

- getText() method can be called from all the Web Elements
- This method returns the visible text if the web element contains any text on it, otherwise it returns nothing

API Syntax:

```
java.lang.String getText()
```

- There is no input parameter for this method
- It returns the visible innerText string of the web element if the web element contains any text on it, otherwise it returns an empty string

Methods to Interact with Web Elements

- getAttribute()
- getText()
- **getCSSValue()**
- getLocation()
- getSize()
- getTagName()

- getCSSValue() method can be called on all the Web Elements
- This method is used to fetch a CSS property value from a web element

API Syntax:

```
java.lang.String getCSSValue(java.lang.String propertyName)
```

- The input parameter is the String value of the CSS property name
- Return type is the value assigned to that property name

Methods to Interact with Web Elements

- getAttribute()
- getText()
- getCSSValue()
- **getLocation()**
- getSize()
- getTagName()

- getLocation() method can be executed on all the Web Elements
- This method is used to get the relative position of the web element rendered on a web page
- The position is calculated relative to the top-left corner of the web page of which (x,y) coordinates are assumed to be (0,0)

API Syntax:

```
Point getLocation()
```

- This method does not take any input parameters
- Return type is a Point class that contains the(x,y) coordinates of the element

Methods to Interact with Web Elements

- getAttribute()
- getText()
- getCSSValue()
- getLocation()
- **getSize()**
- getTagName()

- getSize() method can be called on all the visible Web Elements
- This method returns the width and height of the web element rendered on a web page

API Syntax:

```
Dimension getSize()
```

- This method does not take any input parameters
- Return type is a class instance named Dimension

Methods to Interact with Web Elements

- getAttribute()
- getText()
- getCSSValue()
- getLocation()
- getSize()
- **getTagName()**

- getTagName() method can be called from all the Web Elements
- This method returns the HTML tag name of the web element

API Syntax:

```
java.lang.String getTagName()
```

- The Return type of the method is String
- It returns the tag name of the target element

Perform Actions on Web Elements

sendKeys()

clear()

submit()

- sendKeys() action is applicable for textbox and textarea elements
- It is used to type text into the textbox or a textarea

API Syntax:

```
void sendKeys(java.lang.CharSequence…keysToSend)
```

- The input parameter is CharSequence of text that has to be entered into the web element
- This method does not return anything

Perform Actions on Web Elements

sendKeys()

clear()

submit()

- clear() method is applicable for textbox and textarea elements
- It is used to erase the text entered in a textbox or textarea elements

API Syntax:

```
void clear()
```

- This method does not take any input and does not return any output
- It is executed on the target text-entry element

Perform Actions on Web Elements

sendKeys()

clear()

submit()

- submit() method is used to submit a form of a web page to the server hosting the web application
- This method can be called from any web element in the form

API Syntax:

```
void submit()
```

- This method does not take any input parameters and does not return anything
- However, a NoSuchElementException is thrown when this method is executed on a Web Element that is not present within the form

Checking the Web Element State

- **isDisplayed()**
- isEnabled()
- isSelected()

```
//API Syntax for 'isDisplayed()' method

boolean isDisplayed()

/* Returns "true" if the specified web element is present
 * on the web page and returns "false" if the web element
 * is not present on the web page
*/
```

isDisplayed() method verifies the presence of a web element within the web page

Checking the Web Element State

isDisplayed()

isEnabled()

isSelected()

```
//API Syntax for 'isEnabled()' method

boolean isEnabled()

/* Returns "true" if the specified web element is enabled
 * on the web page and returns "false" if the web element
 * is not enabled on the web page
*/
```

isEnabled() method verifies whether the specified web element is enabled or disabled within the web page. It is mainly used with buttons

Checking the Web Element State

isDisplayed()

isEnabled()

isSelected()

```
//API Syntax for 'isSelected()' method

boolean isSelected()

/* Returns "true" if the specified web element is selected
 * on the web page and returns "false" if the web element
 * is not selected on the web page
*/
```

isSelected() method verifies whether the specified web element is selected or not. It is mainly used with radio buttons, dropdowns and checkboxes

Why do we need Waits in Selenium?

Given a Scenario :

- A Web Application is developed using Ajax and JavaScript
- When the page is loaded by the browser, elements load at different time intervals

Problem:

- Longer page load time
- Partial rendering after page load
- Synchronization between the script and page rendering
- External issues such as low internet and server speed

Solution:

- Manage page load strategy
- Use dynamic waits for web elements
- Set a default polling period as needed wherein any exceptions during the polling period are ignored

Types of Waits in Selenium

- **Implicit**
- Explicit
- Fluent
- pageLoadTimeOut
- setScriptTimeOut

- An Implicit Wait directs the web driver to wait for a certain amount of time before throwing an exception
- An Implicit method is a single line of code, declared in the setup method of the test script
- An Implicit Wait is defined only once and it will remain same throughout the driver object instance

Syntax:

```
driver = new ChromeDriver();
driver.navigate().to("http://xyz.com/");
driver.manage().timeouts().implicitlyWait(TimeOut,TimeUnit.SECONDS);
```

Types of Waits in Selenium

Implicit

Explicit

Fluent

pageLoadTimeOut

setScriptTimeOut

- An Explicit Wait directs the web driver to wait until the time a specific condition is met or wait for the maximum time if a specific condition is not true before throwing an exception
- Explicit Waits are applied for a particular instance only
- Also referred as "Conditional Wait"

Syntax:

```
WebDriverWait wait=new WebDriverWait(WebDriverReference, TimeOut);

WebElement myElement =
   wait.until(ExpectedConditions.visibilityOfElementLocated(By.id
   ("myElement")));
```

Types of Waits in Selenium

- Implicit
- **Explicit**
- Fluent
- pageLoadTimeOut
- setScriptTimeOut

Few Types of Expected Conditions

Expected Condition	Description
elementToBeClickable()	▪ Waits for an element to be clickable meaning the element should be present/displayed/visible or enabled on the screen
textToBePresentInElement()	▪ Waits for an element having a certain string pattern
alertIsPresent()	▪ Waits for an alert box to appear
titleIs()	▪ Waits for a page with a specific title
frameToBeAvailableAndSwitchToIt()	▪ Waits for a frame to be available. Once the frame is available, control switches to it automatically

Types of Waits in Selenium

- Implicit
- Explicit
- **Fluent**
- pageLoadTimeOut
- setScriptTimeOut

- Fluent Wait instance defines the maximum amount of time to wait for a condition, as well as the frequency with which the condition is checked
- Furthermore, the user may configure the wait to ignore specific types of exceptions, such as “NoSuchElementException” when searching for an element on the web page

Syntax:

```
Wait wait=new FluentWait(Web Driver reference)
      .withTimeOut(Duration.ofSeconds(SECONDS))
      .pollingEvery(Duration.ofSeconds(SECONDS))
      .ignoring(Exception.class);
```

Types of Waits in Selenium

Implicit

Explicit

Fluent

pageLoadTimeOut

setScriptTimeOut

- Sets the amount of time to wait for a page to load completely, before throwing an error

Syntax:

```
driver.manage().timeouts().pageLoadTimeOut(TimeOut,TimeUnit.SECONDS);
```

Types of Waits in Selenium

Implicit

Explicit

Fluent

pageLoadTimeOut

setScriptTimeOut

- Sets the amount of time to wait for an asynchronous script to finish execution before throwing an error

Syntax:

```
driver.manage().timeouts().setScriptTimeOut(TimeOut,TimeUnit.SECONDS);
```

What is TestNG?

TestNG

- TestNG is an Automation Testing Framework created to overcome the shortcomings of frameworks like JUnit and NUnit
- It is more powerful and easier to use than JUnit and NUnit
- TestNG can easily create reports of all the test cases, whether passed, skipped or failed

Why TestNG for Selenium?

- Helps structuring Selenium test cases
- Increases script readability
- Enables parallel testing

TestNG Features

Annotations

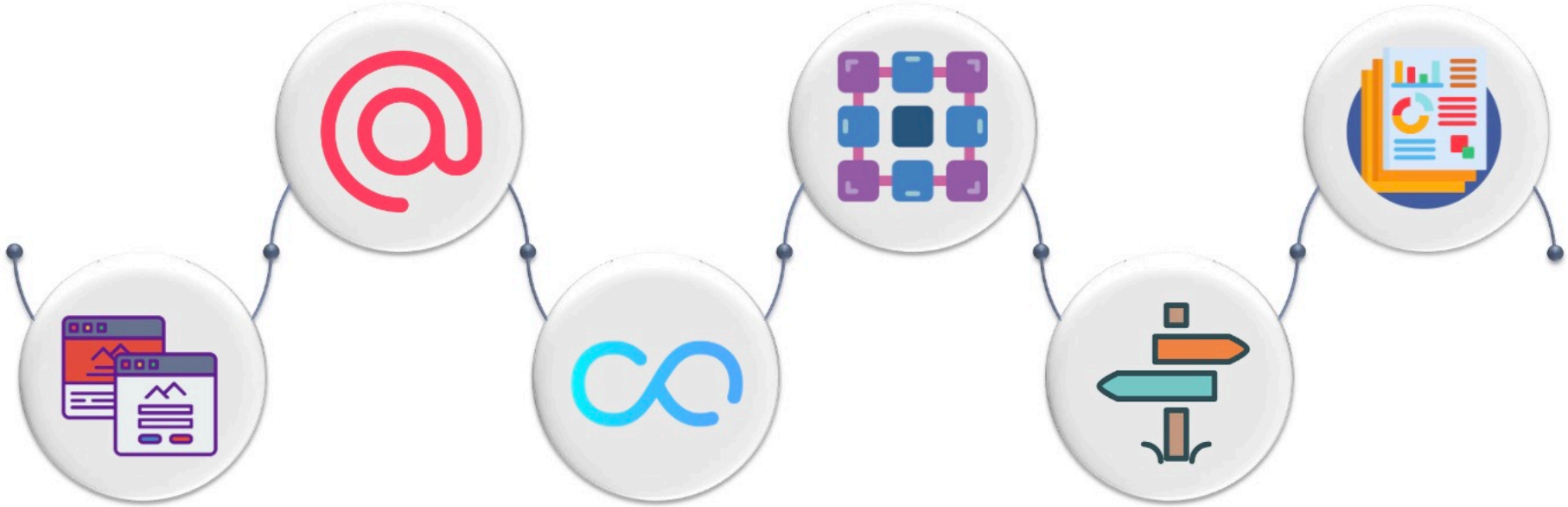
Annotations help remove the clutter and make test scripts more understandable. It also helps give structure to the suite

Grouping

Test cases can be grouped together to run as a collective test suite. Test cases can also be prioritized

Reporting

Can generate reports which include metrics of number of test cases run, test cases failed, and test cases skipped



Cross-browser Testing

Multiple test cases can be executed together across various browsers simultaneously

Tool Integration

Can be easily integrated with third-party tools such as Maven, Jenkins, etc. to extend its functionality

Parameterization

Helps create dynamic test cases. Can be used to parameterize Annotations or even Data Providers

TestNG Annotations

- Annotations define the way code is executed in a test suite
- Annotations start with '@' symbol
- Helps remove clutter and create malleable structure for the test cases

@TestNG

Advantages of Using Annotations

Unrestricted

Restriction for pattern/format are lifted as TestNG specifically looks for annotated methods/classes

Strongly Typed

Typing errors are easily identifiable by the compiler as Annotations are strongly typed

Parameters

Parameterization is possible for annotated methods and classes

Annotations

Annotation	Description
@BeforeSuite	Method runs once before the suite starts executing other testcases
@AfterSuite	Method runs once after all the suite testcases have run
@BeforeClass	Method executes once before the first test is invoked inside the class
@AfterClass	Method executes once after all the tests within the class have run
@BeforeTest	Runs before any other test of a class belonging to the current test-case have run
@AfterTest	Runs after all the tests belonging to the current test-case have run
@BeforeGroups	Runs before any other test belonging to the mentioned group(s) has run
@AfterGroups	Runs after all the tests belonging to the mentioned group(s) have run

Annotations (Contd.)

Annotation	Description
@BeforeMethod	Annotated Method is executed before each method in test suite
@AfterMethod	Annotated Method is executed after each method in test suite
@DataProvider	Labels a method as a data supplier for a test method
@Factory	Returns factory objects that can be used by TestNG as test classes
@Listeners	Defines listeners that can modify the behaviour of TestNG
@Parameters	Used to pass parameters to a @Test method
@Test	Includes a method as part of the test case

Attributes for @Test

Attribute	Description
description	Text string to provide description to the test
timeOut	Maximum time for a test to execute before an exception is generated (in milli seconds)
priority	Sets the order priority for the tests (range -5000 to 5000)
dependsOnMethods	Used when one tests depends upon another for execution
enabled	Used in case if a certain test isn't required to be skipped for a particular run
groups	Defines the group to which the test belongs

TestNG Script Writing Rules

Executing a test case in TestNG involves:

- Write the test case logic using TestNG Annotations
- Create a new testng.xml file and insert the require information about the tests
- Execute the testng.xml file

Basic testng.xml Script Structure

A single suite is defined by a single .xml file. Suite can consist of multiple tests

The <test> tag defines a single test. Tests can contain single or multiple TestNG test classes

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name = "SuiteName">
     <test name = "testName">
          <classes>
               <class name = "TestClassName"/>
          </classes>
     </test>
</suite>
```

The TestNG classes in the business logic are referenced using the <classes> tag

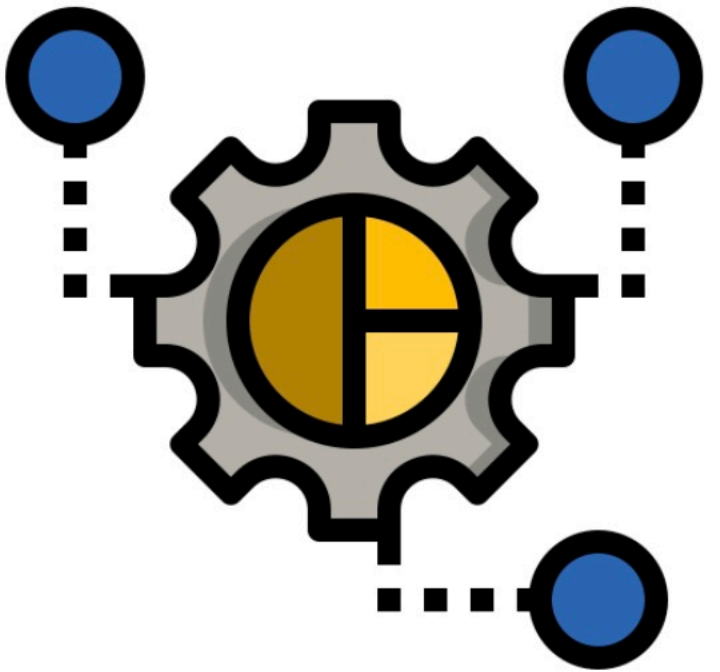
testng.xml Script: Key Points

- Package names can be used instead of classes
- All the classes using TestNG annotations will become a part of the test case
- Individual groups and methods can be included or excluded as a part of the test case
- TestNG preserves the order of the test cases as mentioned in the .xml file
- Random order can be enabled by setting the preserve-order attribute of suite to false

```
<?xml … >
<!DOCTYPE … >

<suite … >
     <test name = "testName">
          <packages>
               …
          </packages>
          <groups>
               …
          </groups>
     </test>
</suite>
```

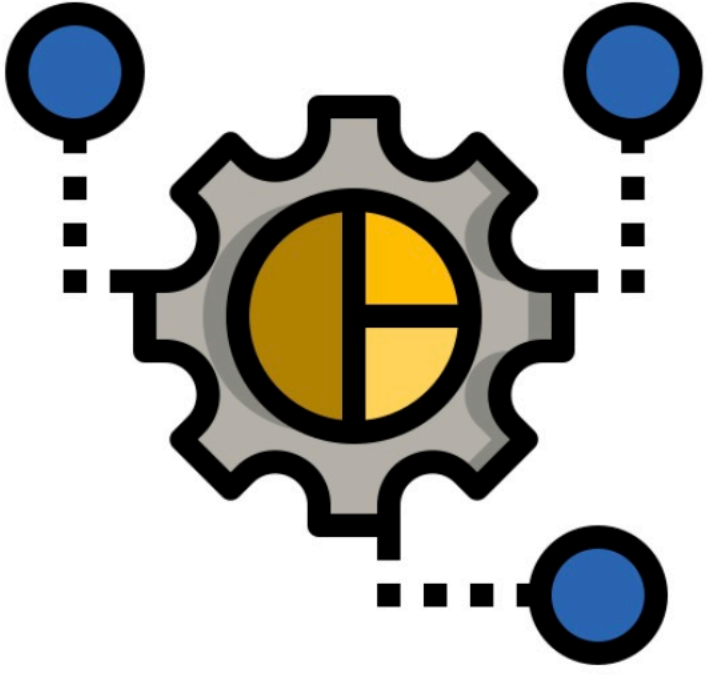
Grouping

- Grouping allows the end user to group the test cases according to their requirement
- It makes the test cases more flexible and modular
- “groups” keyword is added as an attribute to the @Test annotation



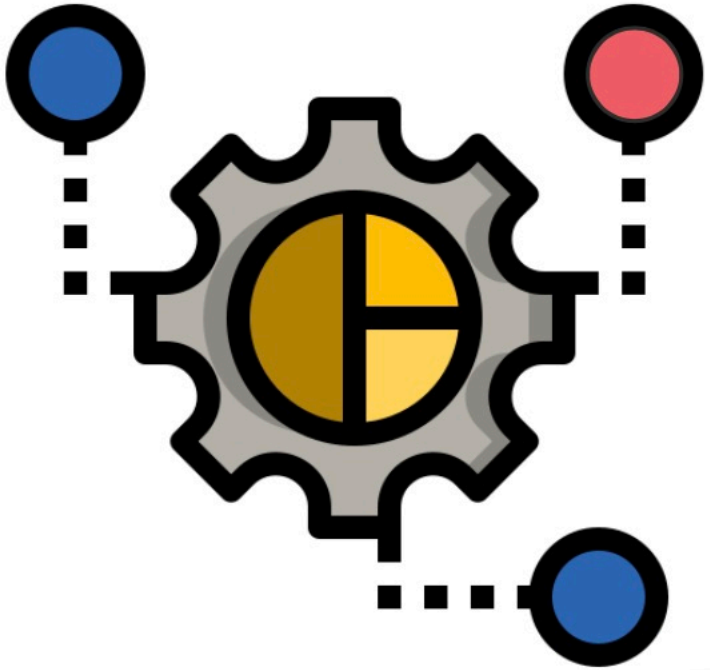
Example: Grouping

```
public class grouping{
      @Test(groups={"group1"})
      public void method1(){
            …
      }

      @Test(groups={"group2"})
      public void method2(){
            …
      }

      @Test(groups={"group1"})
      public void method3(){
            …
      }
}
```



Including and Excluding Groups

- TestNG also enables selective execution of methods using groups
- Groups can be included or excluded in the test case
- It includes/excludes all the methods belonging to a certain group for that execution cycle
- Individual methods can be included/excluded from execution using the same technique



Example: Including and Excluding Groups

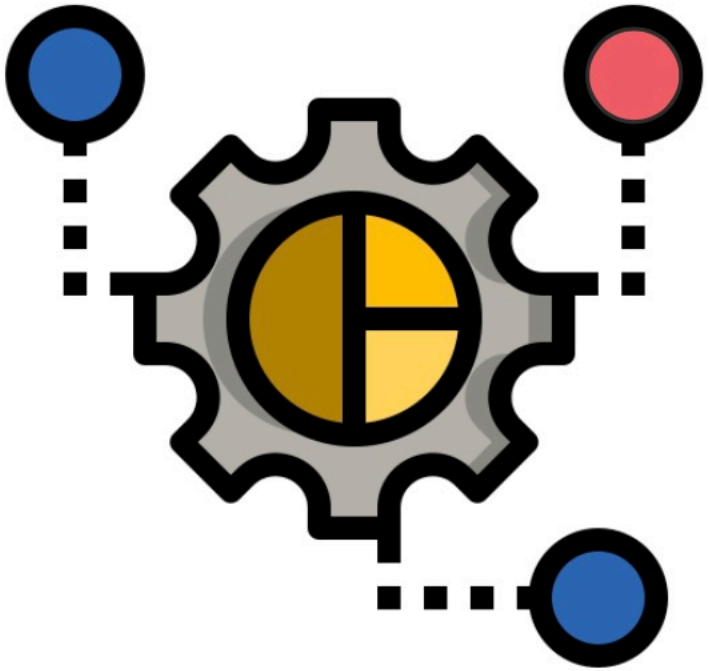
```
public class grouping{
     @Test(groups={"group1"})
     public void method1(){
          …
     }

     @Test(groups={"group1","group2"})
     public void method2(){
          …
     }

     @Test(groups={"group1"})
     public void method3(){
          …
     }
}
```

```
<?xml … >
<!DOCTYPE … >

<suite … >
     <test name = "testName">
          <groups>
            <run>
               <include name="group1"/>
               <exclude name="group2"/>
            </run>
          </groups>
     </test>
</suite>
```

Example: Including and Excluding Groups

In the given example:

- “method1” and “method3” are executed as they belong to “group1”
- “method2” is not executed because it is a part of “group2” which is excluded from execution

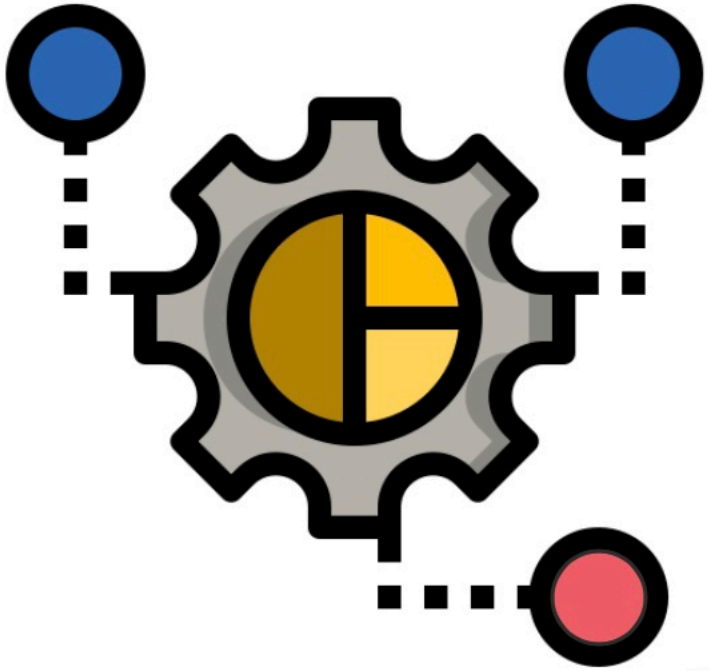


Regex: Including and Excluding Groups

Regular expressions can be used to define included/excluded groups

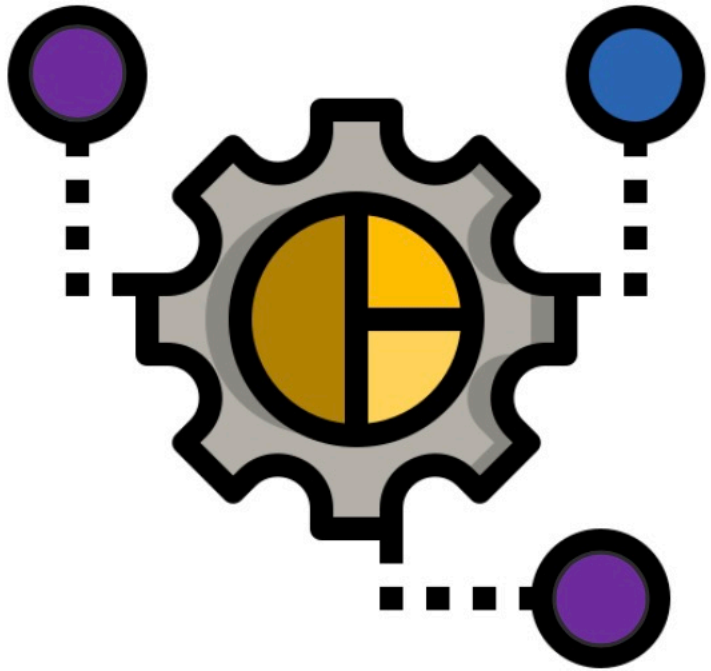
Taking forward the previous example:

- `<include name="group.*">`
- This will include all the groups starting with the string "group"



Default Group

- A default group can be assigned to public methods of a class
- Methods which do not have any groups assigned to them come under the default group
- They are also known as Partial Groups



Default Group

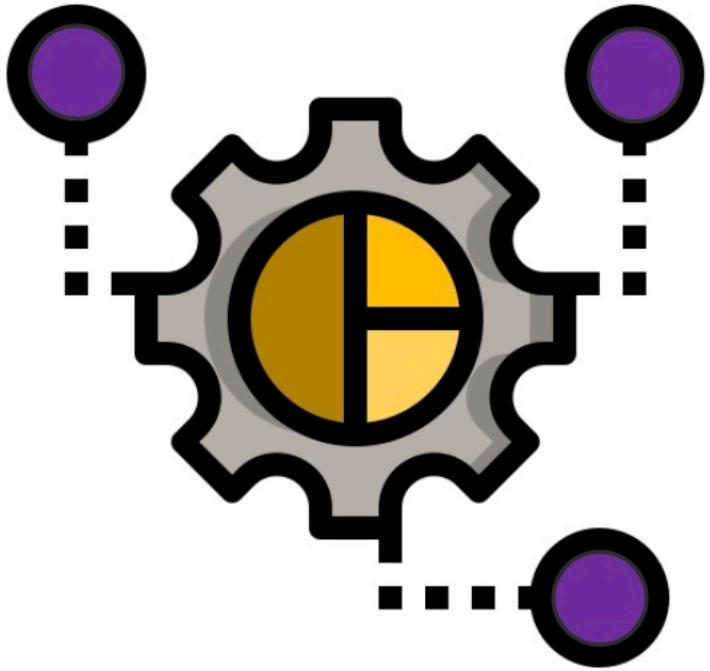
In the given example:

- “method2” and “method3” will be included in the “def-group”
- “method1” will still operate under “group1”

```
@Test(groups={def-group})
public class grouping{
     @Test(groups={“group1”})
     public void method1(){
          …
     }

     @Test
     public void method2(){
          …
     }

     @Test
     public void method3(){
          …
     }
}
```

MetaGroups

- Users can create a group consisting of other groups
- MetaGroups can be created to compile the test logic on a larger level



MetaGroups

```
public class grouping{
     @Test(groups={"group1"})
     public void method1(){
          …
     }

     @Test(groups={"group2"})
     public void method2(){
          …
     }

     @Test(groups={"group3"})
     public void method3(){
          …
     }

}
```

→

```
<?xml … >
<!DOCTYPE … >

<suite … >
     <test name = "testName">
          <groups>
            <define name="metaGroup1">
               <include name="group1"/>
               <include name="group3"/>
            </define>
            <run>
               <include name="metaGroup1"/>
            </run>
          </groups>
     </test>
</suite>
```

Assertions

The Assertion method ensures that the state of the application is as it is supposed to be. There are two types of assertions

Assertions: Hard Assert

Hard Assert 1

Soft Assert 2

- When a **hard assert** fails, test execution is aborted and the compiler throws an exception
- The test case is immediately stopped and marked as failed
- All the assertions are hard asserts unless used with a soft assert object

Assertions: Soft Assert

Hard Assert 1

Soft Assert 2

- In case of **soft assert,** the statements after the assertion are executed regardless of result of the assertion
- **Soft Assertions** does not fail the test case and instead shows test case to be successful
- **Soft Assertions** are used in cases where the code after the current test case is required to run

Assertions: Common Assert Commands

- **assertEquals**
- assertNotEquals
- assertNull
- assertNotNull
- assertTrue
- assertFalse

- Compares the actual outcome with the expected outcome
- Consists of three parameters including the actual outcome, expected outcome and a failure message(optional)
- If the outcomes are **not equal** the execution stops and the message is displayed

Syntax: `assertEquals(Object actual, Object expected, message);`

Assertions: Common Assert Commands

- assertEquals
- **assertNotEquals**
- assertNull
- assertNotNull
- assertTrue
- assertFalse

- It compares the actual outcome with expected outcome and looks for inequality
- Consists of three parameters including the actual outcome, expected outcome and a failure message(optional)
- If the outcomes are **equal** the execution stops and the message is displayed

Syntax: `assertNotEquals(object actual, Object expected, message);`

Assertions: Common Assert Commands

- assertEquals
- assertNotEquals
- **assertNull**
- assertNotNull
- assertTrue
- assertFalse

- It checks whether the object being passed is **Null**
- Two parameters are used: the first one being the object to be tested and the second is the message(optional) parameter
- The test case **fails** if the object is **not Null**

Syntax: `assertNull(object, message);`

Assertions: Common Assert Commands

- assertEquals
- assertNotEquals
- assertNull
- **assertNotNull**
- assertTrue
- assertFalse

- It checks whether the object being passed is **not Null**
- Two parameters are used: the first one being the object to be tested and the second is the message(optional) parameter
- The test case **fails** if the object is **Null**

Syntax: `assertNotNull(object, message);`

Assertions: Common Assert Commands

- assertEquals
- assertNotEquals
- assertNull
- assertNotNull
- **assertTrue**
- assertFalse

- assertTrue helps the user to assert the given condition is true
- Two parameters are passed; first one being the condition and the second one is the message(optional)
- If the condition is **false** the test case fails and the optional message is displayed

Syntax: `assertTrue(condition, message);`

Assertions: Common Assert Commands

- assertEquals
- assertNotEquals
- assertNull
- assertNotNull
- assertTrue
- **assertFalse**

- assertFalse helps the user to assert the given condition is false
- Two parameters are passed; first one being the condition and the second one is the message(optional)
- If the condition is **True** the test case fails and the optional message is displayed

Syntax: `assertFalse(condition, message);`

Soft Asserts

- Soft asserts allow us to continue test execution even though the assertion has failed
- In order to use soft assert, a `SoftAssert` object has to be invoked
- SoftAsserts keep all the assertions found during the execution of @Test method
- The assertAll() method can be used at the end of the test to fail the test

Example:

```
SoftAssert sa = new SoftAssert();
sa.assertTrue(object,"soft assert example");
```

Parameterization in TestNG

Parameterization helps users write dynamic test scripts by passing variable parameters through annotations

Parameterization in TestNG

- Parameters can be added to any method having @Test @Before/After or @Factory annotation
- Any number of parameters can be set for a test method
- Parameters are scoped variables
- Parameters defined in <test> take precedence over parameters defined in <suite>

TestNG

Types of Parameterization in TestNG

Parameters with Annotations

Parameters with Data Providers

Annotations: Invoking Parameters

```
public class parameterDemo{

     @Parameters({"par1","par2"})
     @Test
     public void method1(String pV1, String pV2){
          System.out.println(pV1 + pV2)
     }

}
```

Annotations: Defining Parameters in testng.xml

```
<?xml … >
<!DOCTYPE … >

<suite … >
     <parameter name="par1" value="value1"/>
     <parameter name="par2" value="value2"/>
     <test>
     </test>
</suite>
```

Parallel Testing in TestNG

- TestNG enables testers to execute multiple test suites/tests together
- It can easily run in multi-threaded mode
- This allows faster execution of multiple test-cases, reducing the execution time considerably

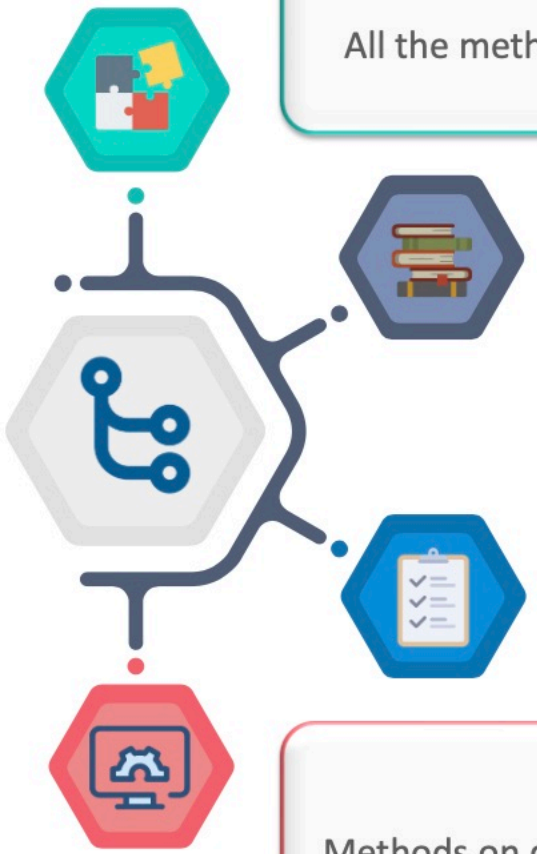
Attributes: Parallel Testing

- The `thread-count` attribute specifies the number of threads to be allocated for execution
- `threadPoolSize` attribute defines the number of different threads to invoke a method
- `timeOut` attribute ensures that a single thread doesn't block the current thread

Parallel Attribute in TestNG

The parallel attribute in the <suite> tag defines the scope of parallel testing. The attribute can take one of the following values:

Parallel Attribute in TestNG (Contd.)



Methods

All the methods are executed in separate threads but maintain the specified order

Classes

All the methods belonging to a class run on a single thread, but separate classes get separate threads

Tests

All methods part of a test case run on a single thread, but different tests get separate threads

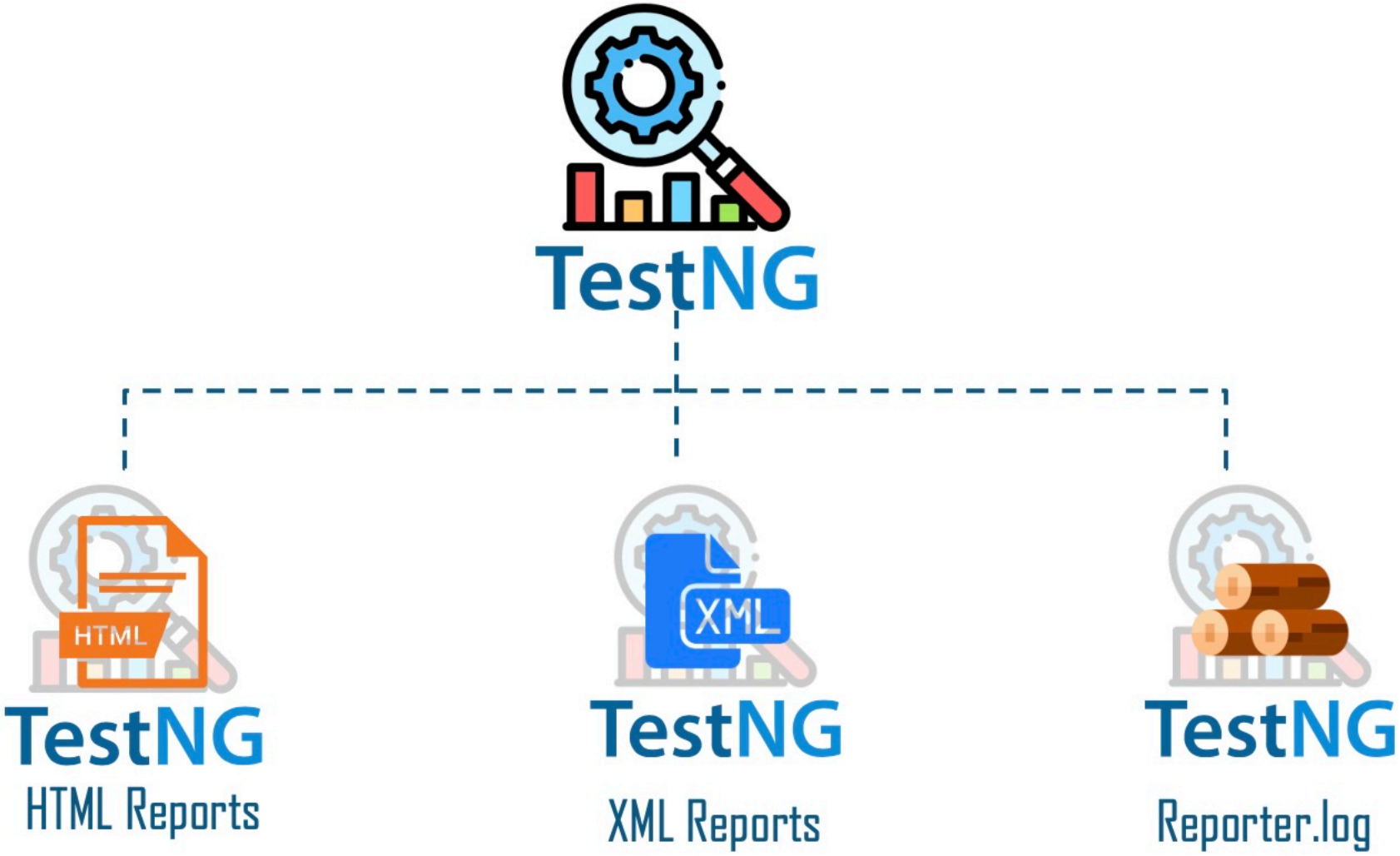
Instances

Methods on different instances run on different threads

TestNG Reporting

- TestNG generates HTML reports after the test-cases are executed
- Reports are helpful in presenting the status of the overall project
- The reports are only generated for methods annotated as @Test

Types of TestNG Reports



Types of TestNG Reports: HTML Reports

- By default, TestNG creates an emailable-report.html file in the test-output directory
- The index.html file contains a more detailed summary of the generated report

TestNG

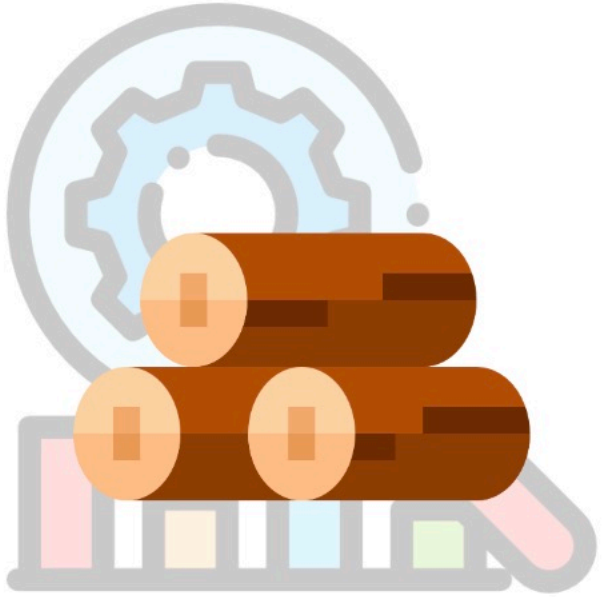
Types of TestNG Reports: XML Report

- TestNG offers xml-report capturing
- These reports capture TestNG specific data which other methods do not capture
- The reporter-api is injected using -reporter flag via the command line

TestNG

Reporter Class

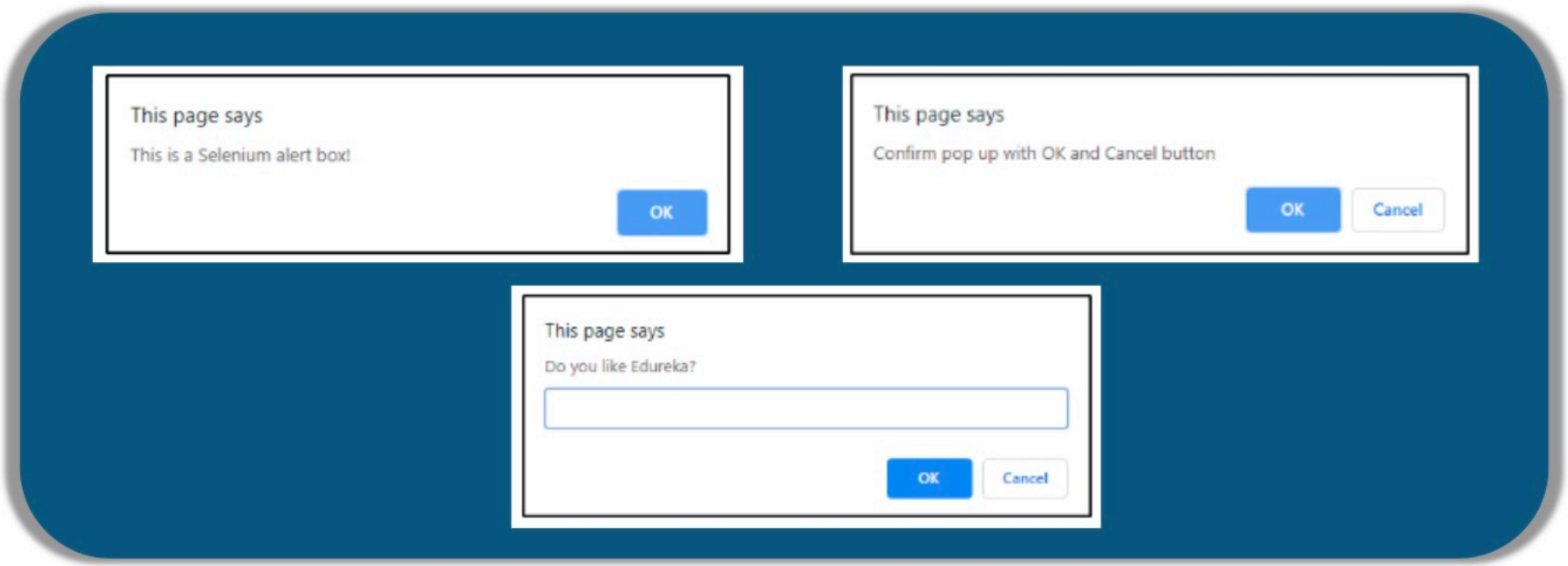
- Users can add logs to mark events in the console window during a test
- Two types of logging can be done:
  - Low-level: Logs are produced for each action performed during a test
  - High-level: Logs are captured for main-events only



TestNG

What are Alerts?

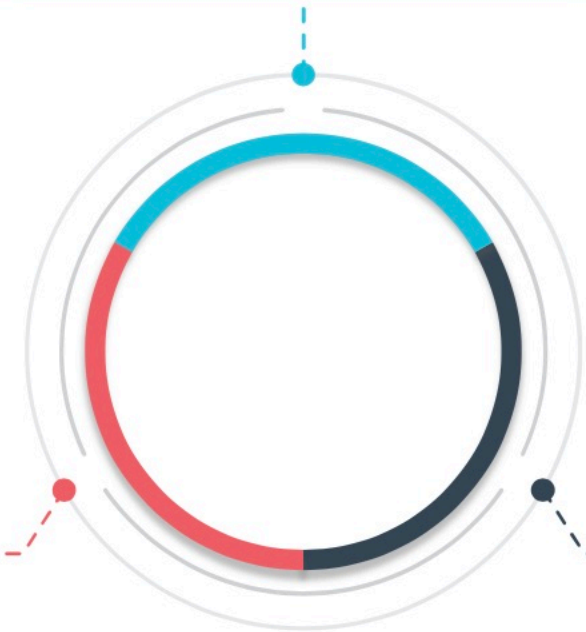
- Alerts are small pop ups or message boxes that takes the users focus away from the current browser and compels to read the alert message
- It is an on-screen notification to provide the user with some information or ask for an action



Types of Alerts

Simple Alert

Displays some information or warning on the screen



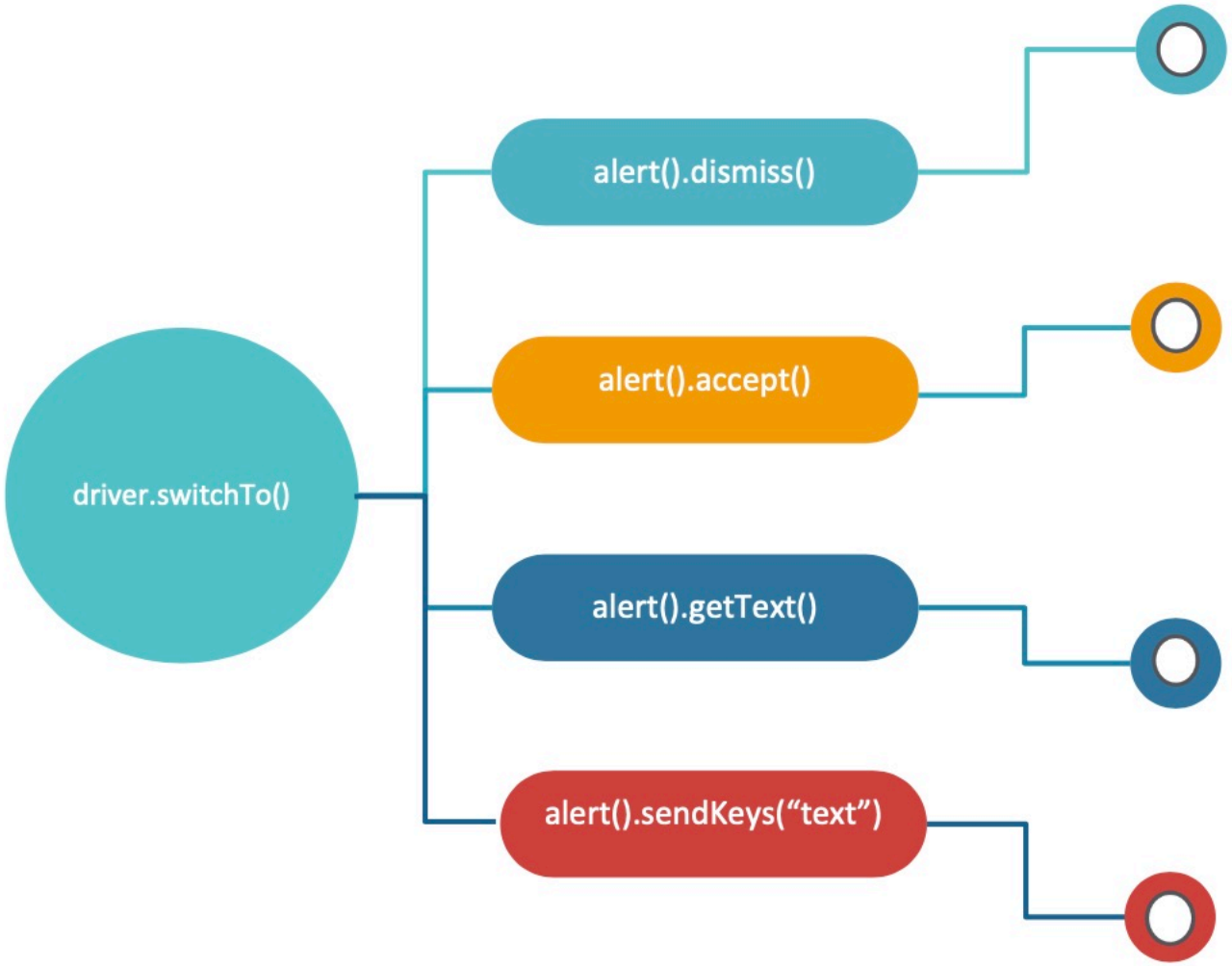
Prompt Alert

Requests some information from the user

Confirmation Alert

Requests permission to do some type of operation

Alert Interface Methods



This method is called when the 'Cancel' button is clicked in the alert box

This method is called when the 'OK' button is clicked in the alert box

This method is called to capture the alert message

This method is called when you want to send some data to the alert box

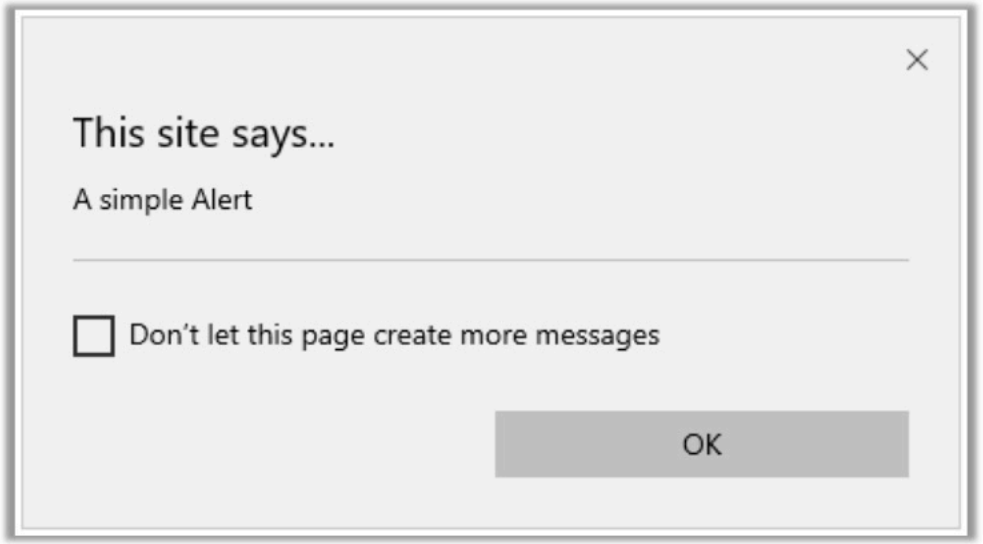
Types of Alerts

- Simple
- Prompt
- Confirmation

Simple Alert

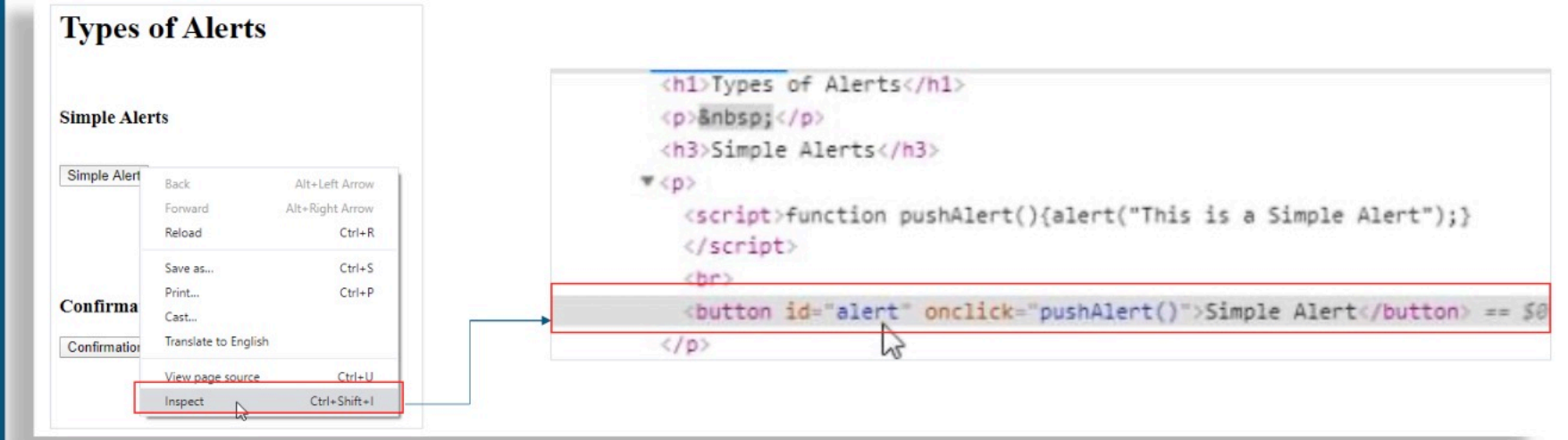
- This type of alert is mainly used to display some information or warning on the screen
- This type of alert comes with an option to accept the alert

```
alert.accept(); //Accept the alert
```



Types of Alerts

- Simple
- Prompt
- Confirmation




```
//Code snippet to handle a Simple Alert

driver.get("D:\\alerts.html");

driver.manage().window().maximize();
driver.findElement(By.id("alert")).click();

Alert simplealert = driver.switchTo().alert();

String alertMessage = simplealert.getText();
System.out.println(alertMessage);
simplealert.accept();
```

Types of Alerts

Simple

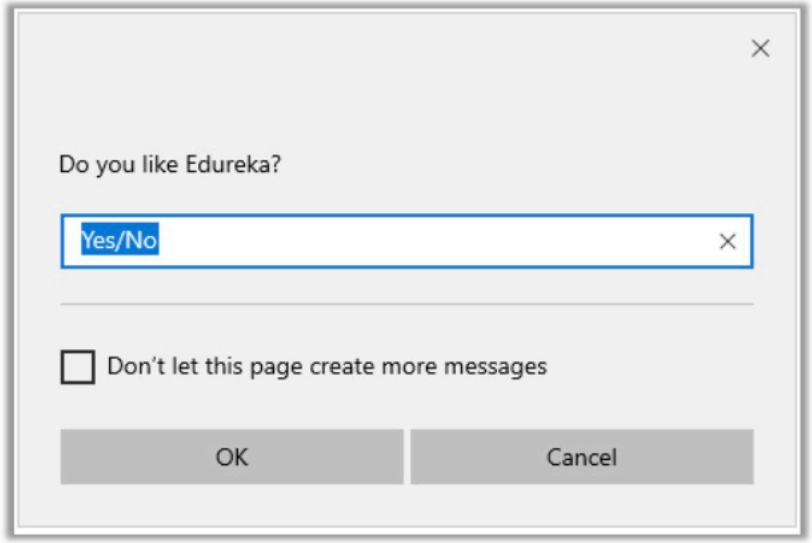
Prompt

Confirmation

Prompt Alert

- In this type of alert, there exists an option to add a text field to the Alert box
- The sendKeys() method is used to type some text in the Prompt Alert box

```
alert.sendKeys("Accept the Alert");
alert.accept();      //Accept the alert
alert.dismiss();     //Dismiss the alert
```



Types of Alerts

Simple

Prompt

Confirmation

```
//Code snippet to handle a Prompt Alert

driver.get("D:\\alerts.html");

driver.manage().window().maximize();
driver.findElement(By.id("prompt")).click();

Alert promptalert = driver.switchTo().alert();

String alertMessage = promptalert.getText();
System.out.println(alertMessage);
promptalert.sendKeys("Accept the alert");
promptalert.accept();
```

Types of Alerts

Simple

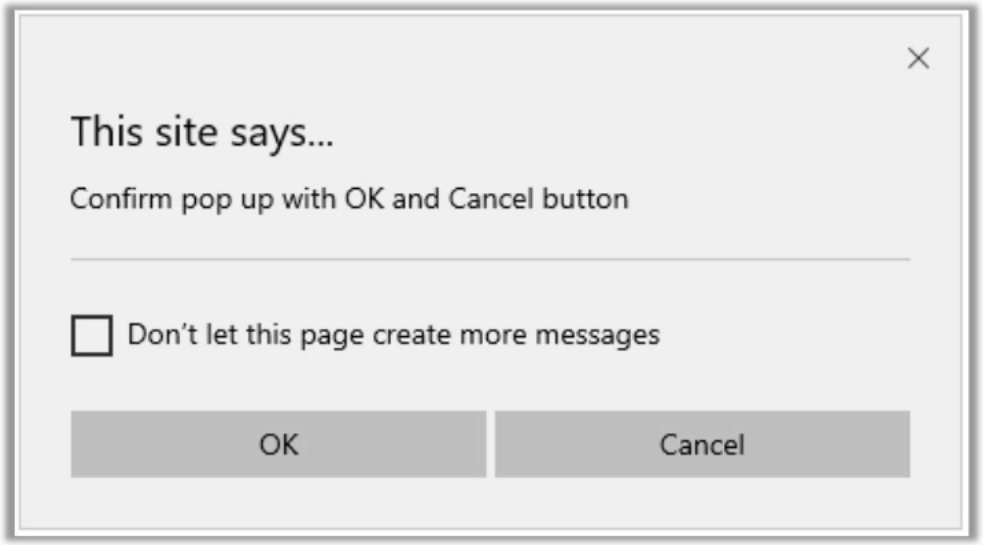
Prompt

Confirmation

Confirmation Alert

- This type of alert comes with an option to accept or dismiss the alert

```
alert.accept(); //Accept the alert
alert.dismiss(); //Dismiss the alert
```



Types of Alerts

Simple

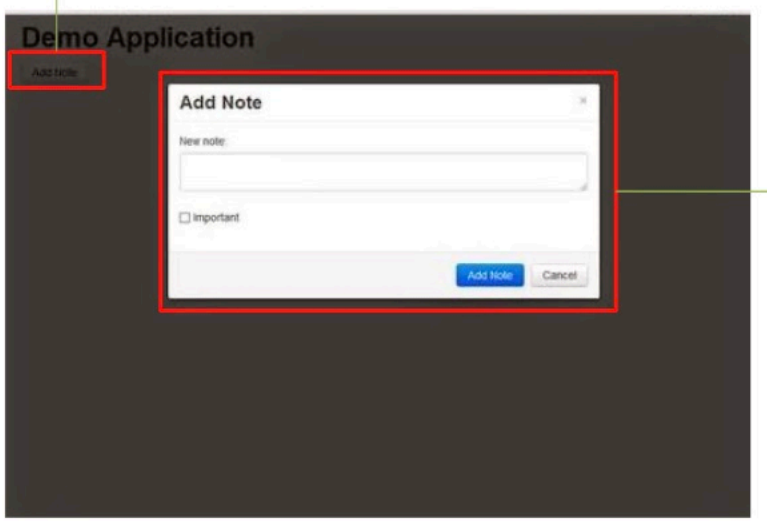
Prompt

Confirmation

```
//Code snippet to handle a Confirmation Alert

driver.get("D:\\alerts.html");

driver.manage().window().maximize();
driver.findElement(By.id("confirm")).click();

Alert confirmalert = driver.switchTo().alert();

String alertMessage = confirmalert.getText();
System.out.println(alertMessage);
confirmalert.dismiss();
```

What is a Modal Dialog Box?

A Modal Dialog Box is a dialog window that forces the user to interact with it before returning to the parent application



Click the "Add Note" button, which in turn opens a container with a space to add a note

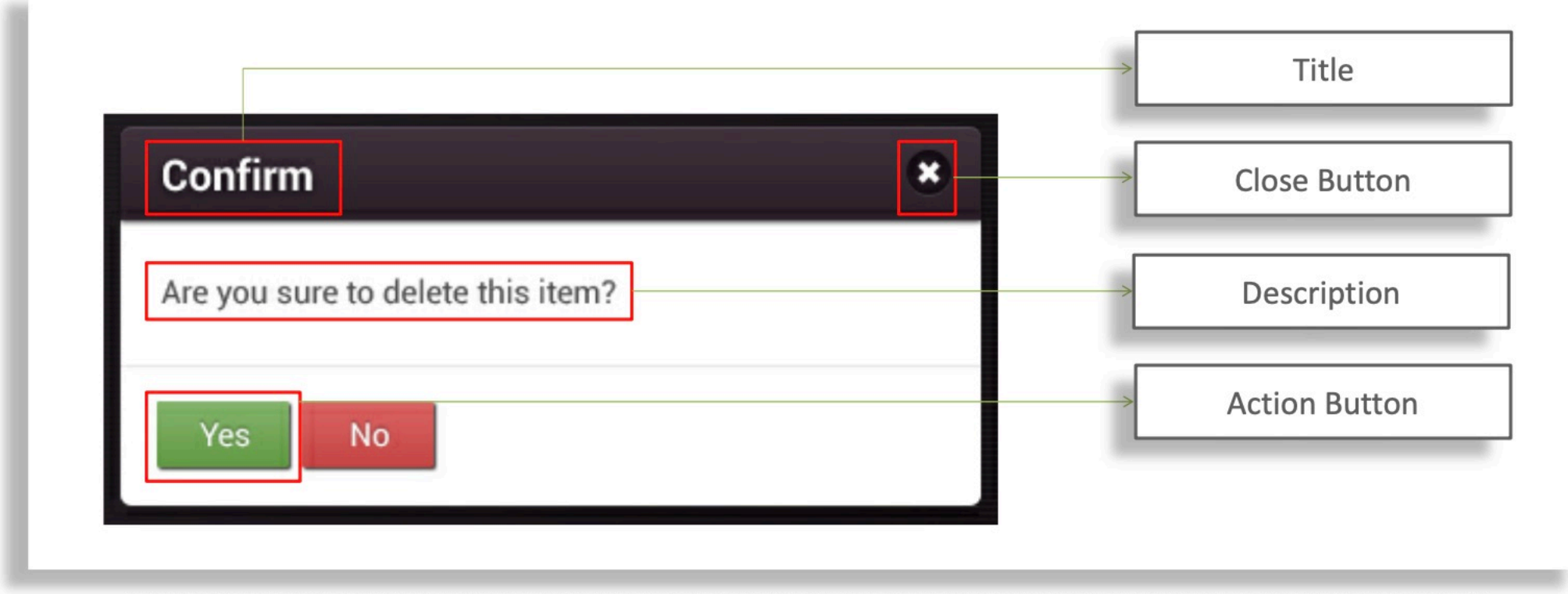
Modal Dialog Box

Messages are not locked, and you can click anywhere to dismiss the container

Components of a Modal Dialog Box

A Modal is an HTML element that is displayed above the current page

It is used to perform a task or a group of tasks



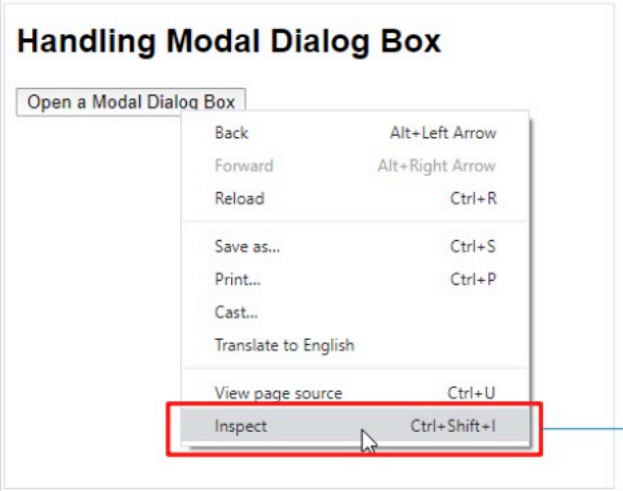
A user cannot interact with the main page until the modal is closed

How Modal Poses Problem for Test Automation Script?

If a Modal is displayed consistently on a webpage → It is easier for the Test Automation Script to interact with it

If a Modal is displayed randomly on a web page → Test Automation Script needs to deal with the Modal when displayed or not displayed

Handling Modal Dialog Box




```
<!doctype html>
<html>
▶ <head>…</head>
▼ <body>
    <h2>Handling Modal Dialog Box</h2>
    <!-- Trigger/Open The Modal -->
    <button id="myBtn">Open a Modal Dialog Box</button> == $0
    <!-- The Modal -->
```

On click of the button, a modal dialog box opens with some text in it

```
  <!-- The Modal -->
▼ <div id="myModal" class="modal" style="display: block;">
    <!-- Modal content -->
  ▼ <div class="modal-content">
      <span class="close">×</span> == $0
      <p>Some text in the Modal..</p>
    </div>
```

Handling Modal Dialog Box (Contd…)

```
//Code snippet to handle Modal Dialog Box using Selenium Web Driver

driver.get("D:\\modal.html");
driver.manage().window().maximize();

driver.findElement(By.id("myBtn")).click();

String s1=driver.findElement(By.className("modal-content")).getText();

System.out.println("Modal Dialog Box Content:"+s1);

driver.findElement(By.className("close")).click();
```

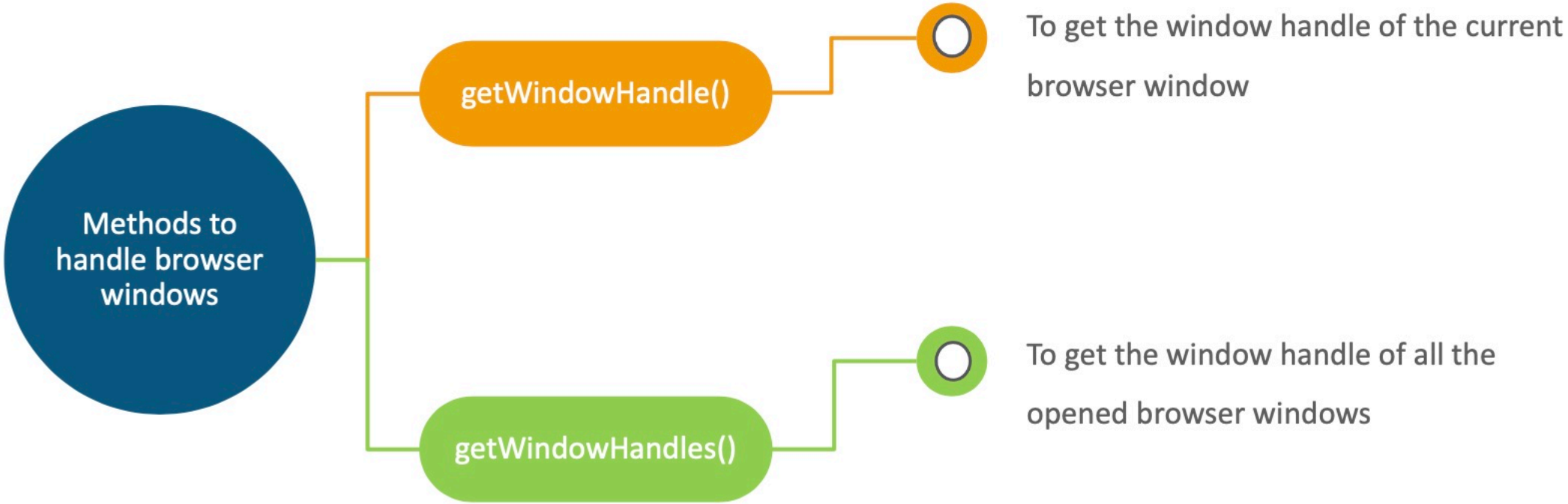
Output:

```
Modal Dialog Box Content:x
Some text in the Modal..
```

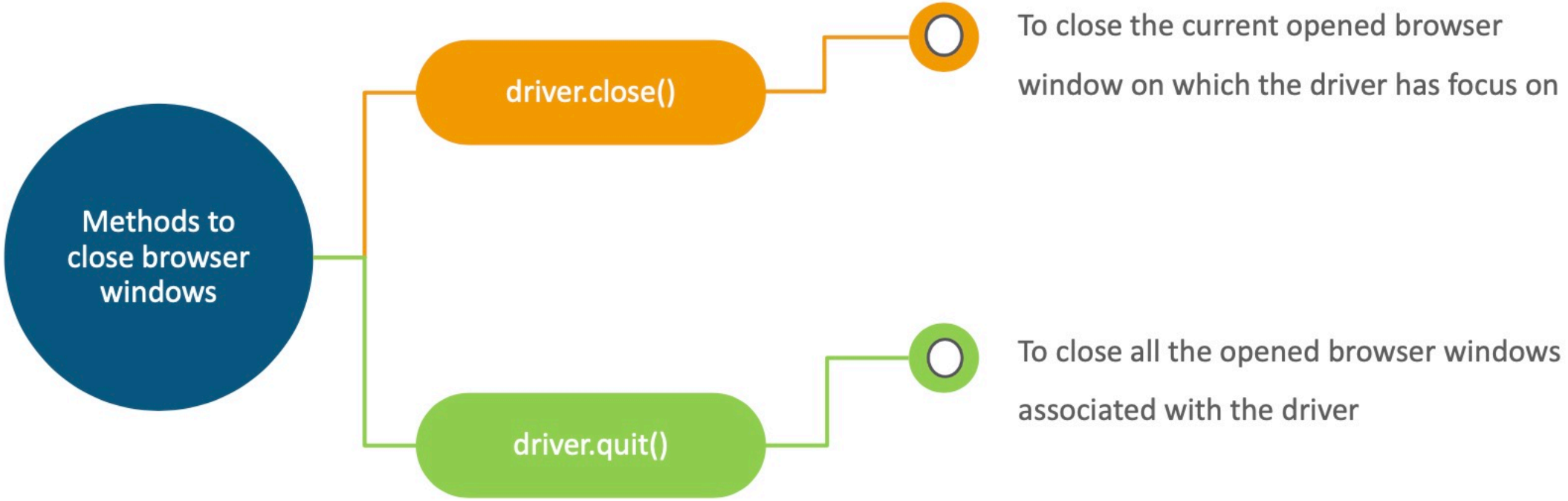
Handling Multiple Browser Windows

- Few web applications have multiple windows
- Once the WebDriver object is instantiated, Selenium WebDriver assigns an alphanumeric id to each window
- The Alphanumeric id is known as a Window Handle which is a unique identifier that uniquely identifies the browser window

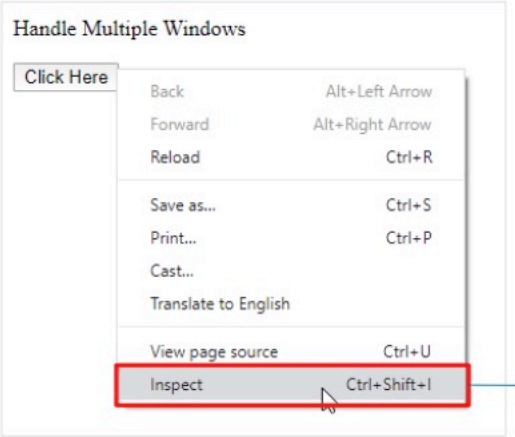
WebDriver Methods to Handle Browser Windows



WebDriver Methods to Close Browser Windows



Handling Multiple Browser Windows




```
<!doctype html>
<html>
▶ <head>…</head>
▼ <body>
    <p>Handle Multiple Windows</p>
    <button id="btn" onclick="myFunction()">Click Here</button> == $0
  ▼ <script>

      function myFunction() {

          var myWindow = window.open("http://newtours.demoaut.com/ ", "test",
      "width=200,height=100");

          }

    </script>
  </body>
</html>
```

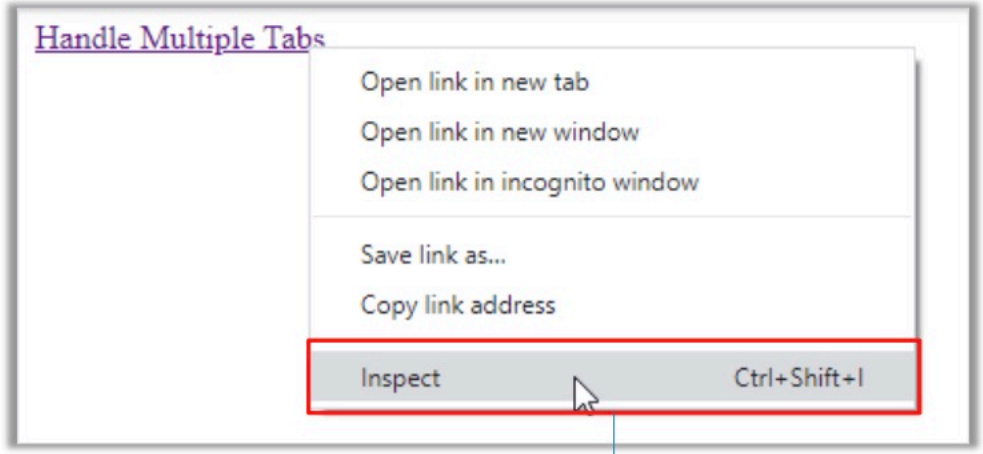
Handling Multiple Browser Windows (Contd…)

```
//Code snippet to handle Multiple Browser Windows
driver.get("D:\\multiplewin.html");
driver.findElement(By.id("btn")).click();

Set<String> handler=driver.getWindowHandles();
Iterator<String> itr=handler.iterator();

String MainWindowId=itr.next();
System.out.println("Parent Window Id: "+MainWindowId);

String ChildWindowId=itr.next();
System.out.println("Child Window Id: "+ChildWindowId);

driver.switchTo().window(ChildWindowId);
System.out.println("Child Window Title is: "+driver.getTitle());
driver.close();

driver.switchTo().window(MainWindowId);
System.out.println("Main Window Title is :"+driver.getTitle());
driver.close();
```

Output:

```
Parent Window Id: CDwindow-A92EA3DEE00E8A71C61A92E49CF22E50
Child Window Id: CDwindow-E6BFEDE88B98E6CA3FD4943AC707B24B
[1579691463.862][WARNING]: Timed out connecting to Chrome, retrying..
Child Window Title is: Welcome: Mercury Tours
Main Window Title is :Main Page
```

Handling Multiple Browser Tabs




```
<!doctype html>
<html>
  <head></head>
▼ <body>
    <a href="http://newtours.demoaut.com/mercurywelcome.php" target="_blank">
    Handle Multiple Tabs</a> == $0
  </body>
</html>
```

Handling Multiple Browser Tabs (Contd…)

```
//Code snippet to handle Multiple Browser Tabs

driver.get("D:\\multipletabs.html");
String MainWindow=driver.getWindowHandle();
System.out.println("Main Window Handle: "+MainWindow);

WebElement clickelement=driver.findElement(By.linkText("Handle Multiple Tabs"));
for(int t=0;t<3;t++)
        {
        clickelement.click();
        Thread.sleep(2000);
        }
        //Handle all opened windows
Set <String> s1=driver.getWindowHandles();

Iterator <String> i=s1.iterator();

for(String handle:s1)
{
System.out.println("Window Handle: "+handle);
}
```

Handling Multiple Browser Tabs (Contd…)

```
//Code snippet to handle Multiple Browser Tabs

while(i.hasNext())
{
String childwindow=i.next();
 if(!MainWindow.equalsIgnoreCase(childwindow))
            {
                        // Switching to Child window
                        driver.switchTo().window(childwindow);
                        driver.findElement(By.name("userName"))
                        .sendKeys("ak@gmail.com");

                        WebElement closeelement=driver.findElement(By.name("login"));

                        closeelement.click();
                        driver.close();
                        Thread.sleep(3000);

            }
        }
                        // Switching to Parent window i.e Main Window
                        driver.switchTo().window(MainWindow);
                        driver.quit();
```

What is JavaScriptExecutor?

- JavaScriptExecutor is an Interface that assists in executing JavaScript using Selenium Web Driver
- JavaScriptExecutor is used to perform a desired operation on a web element

Browsers

Injects JavaScript into the browsers

JavaScriptExecutor

Syntax - JavaScriptExecutor

```
//Syntax to import JavaScriptExecutor

import org.openqa.selenium.JavascriptExecutor;
JavascriptExecutor js=(JavascriptExecutor)driver;

/*executeScript() method executes JavaScript in the context
 **of the currently selected frame or window in Selenium
 */
js.executeScript(Script,Arguments);

/*Script: JavaScript to be executed.
**Arguments: Arguments to the Script (optional), may be empty
**This method returns boolean, long, String, List, Web element,
OR NULL
*/
```

JavaScript Methods for Scrolling a Web Page

scrollBy()

scrollTo()

scrollIntoView()

- **scrollBy()** method is used to scroll the web page to a specific number of pixels

Syntax:

- Following method is used to scroll the web page by the specific number of pixels

```
executeScript("window.scrollBy(xnum,ynum)");
```

xnum: is the number at x-axis. Scrolls horizontally towards the left if the number is positive and towards the right if the number is negative

ynum: is the number at y-axis. Scrolls vertically down if the number is positive and up if the number is negative

JavaScript Methods for Scrolling a Web Page

scrollBy()

scrollTo()

scrollIntoView()

- **scrollTo()** method is used to scroll till the end of the web page

Syntax:

- Following method is used to scroll till the end of the web page

```
executeScript("window.scrollTo(0,document.body.scrollHeight)");
```

document.body.scrollHeight returns the complete height of the web page

JavaScript Methods for Scrolling a Web Page

scrollBy()

scrollTo()

scrollIntoView()

- **scrollIntoView()** method is used to scroll the web page until the specified element is in full view

Syntax:

- Following method is used to scroll until the specified element is visible on the web page

```
executeScript("arguments[0].scrollIntoView();",Element );
```

arguments[0] :First index of the web page starting at 0

Element: Locator on the web page

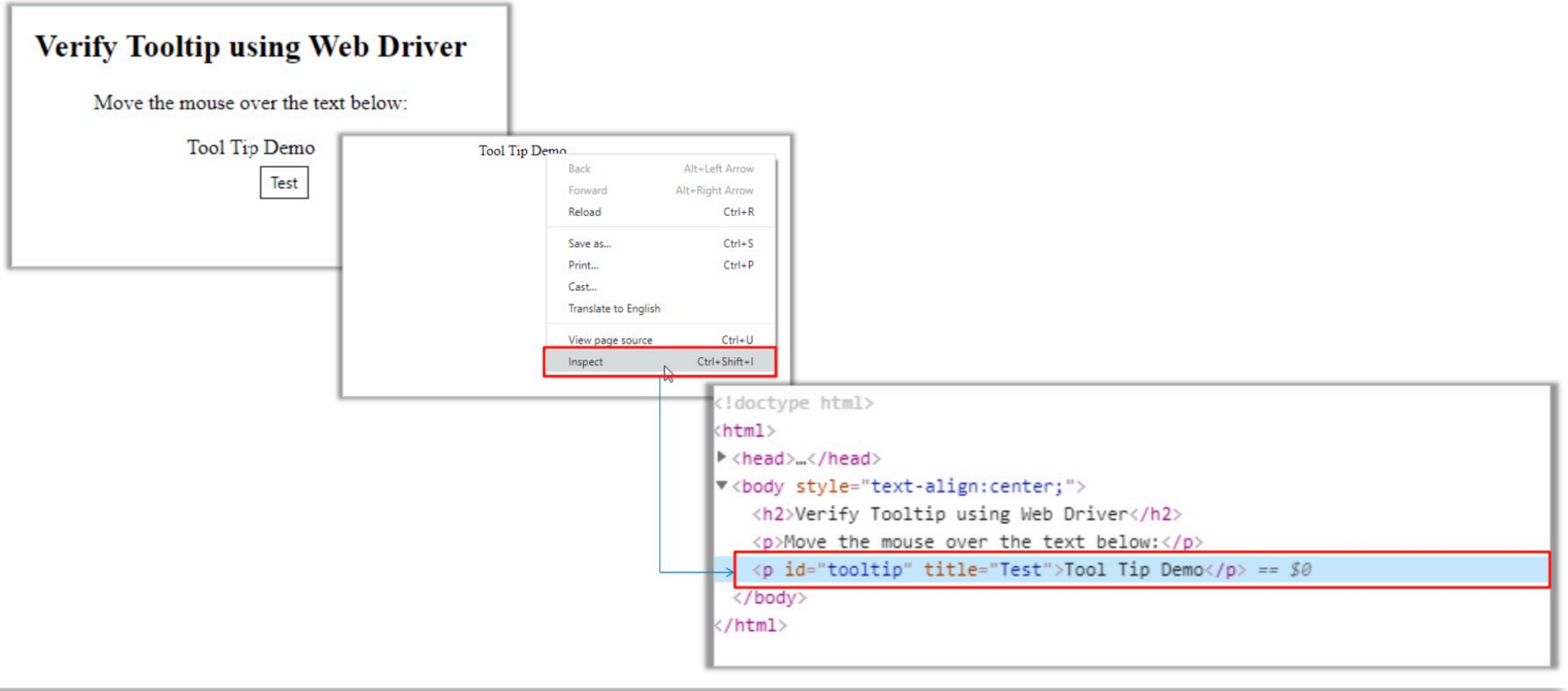
Scenarios: Scrolling on a Web Page

```
/*Scenario 1: Scroll down the web page by 100 pixels through executeScript
**scrollBy() method scrolls the web page to 100 pixels
*/

      JavascriptExecutor js=(JavascriptExecutor)driver;
      js.executeScript("window.scrollBy(0,100)","");
      Thread.sleep(3000);

/*Scenario 2: Scroll down till the end of the web page
**scrollTo() method scrolls till the end of the web page
*/

      js.executeScript("window.scrollTo(0,document.body.scrollHeight)");
      Thread.sleep(3000);
```

Scenarios: Scrolling on a Web Page (Contd…)

```
/*Scenario 3: Scroll up a web page by pixels
**scrollBy() method is used to scroll the web page to 100 pixels
*/
      js.executeScript("window.scrollBy(0,-100)","");
      Thread.sleep(3000);

/*Scenario 4: Scroll down the web page by visibility of element on the web page
**scrollIntoView() method scrolls the web page until the specified element is in full view
**arguments[0] means the first index of page starting at 0, element is the locator on the page
*/
      WebElement element=driver.findElement(By.className("footer"));
      js.executeScript("arguments[0].scrollIntoView();",element);
      Thread.sleep(3000);
```

Verify ToolTip using Selenium WebDriver



Verify Tooltip using Selenium WebDriver (Contd…)

```
//Code snippet to verify ToolTip using getAttribute() method

WebElement element=driver.findElement(By.id("tooltip"));
String expectedToolTip="Test";
String actualTooltip = element.getAttribute("title");

//Check whether the tooltip's value is as expected
System.out.println("Actual Title of Tool Tip is : "+actualTooltip);

if(actualTooltip.equals(expectedToolTip)) {
      System.out.println("Test Case Passed");
      Thread.sleep(3000);
    }
    else
    {
      System.out.println("Test Failed");
    }
```

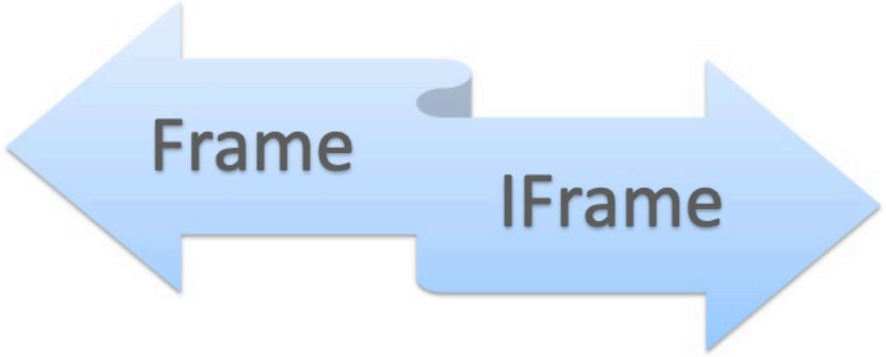
Output:

```
Actual Title of Tool Tip is : Test
Test Case Passed
```

What is an IFrame?

IFrame stands for Inline Frames

A web page embedded within another web page or an HTML document embedded within another HTML document is known as a **Frame**



An **IFrame** is used to embed content from an outside source into a Web Page

How to Identify an IFrame?

Right-click on the web page, click on the “View Page Source” option and search for the <iframe> tag

```
<div class="box">
<iframe name="iframe1" id="IF1" height="600" width="400"
src="http://www.test.co"></iframe>
</div>

<div class="box">
<iframe name="iframe2" id="IF2" height="600" width="400"
src="http://www.test.co"></iframe>
</div>
```

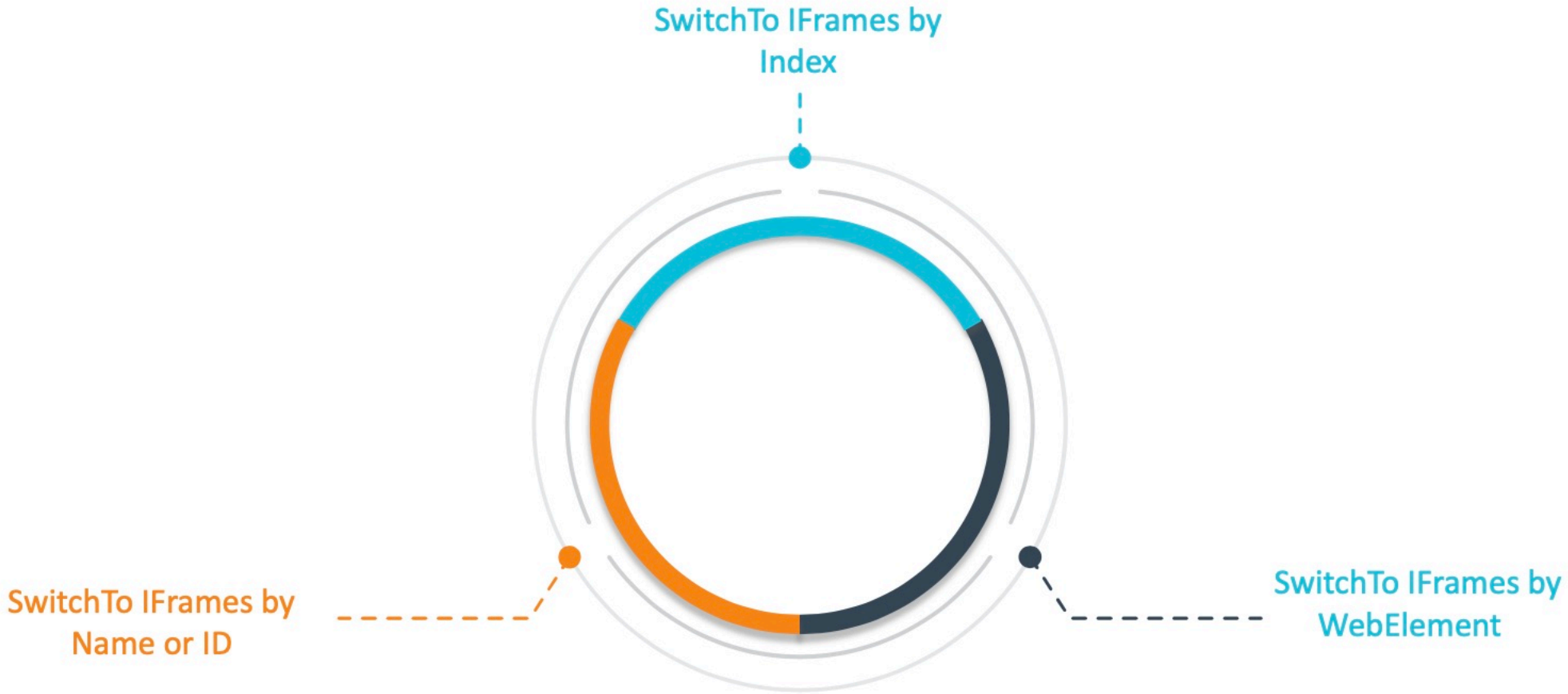
The <iframe> tag specifies an Inline Frame

The following code snippet identifies the total number of IFrames

```
int size = driver.findElements(By.tagName("iframe")).size();
```

Switching to IFrames using Selenium WebDriver

We can switch to the elements in IFrames through the following ways:



Switch to the IFrame By Index

By Index

By Name or ID

By WebElement

```
//Syntax for switching to the IFrames By Index

/*Switch to the IFrame with index 0*/
driver.switchTo().frame(0);

/*Switch to the IFrame with index 1*/
driver.switchTo().frame(1);
```

Index of an IFrame is the position at which it occurs in an HTML Page

Index of an IFrame starts with 0

Switch to the IFrame By Name or ID

By Index

By Name or ID

By WebElement

```
//Syntax for switching to the IFrames by Name or ID

/*Switch to the IFrame By Name*/
driver.switchTo().frame("iframe1");

/*Switch to the IFrame By ID*/
driver.switchTo().frame("IF1");
```

Name and ID are the attributes of an IFrame through which we can switch to it

Switch to the IFrame By WebElement

By Index

By Name or ID

By WebElement

```
//Syntax for switching to the IFrames by WebElement

/*Locate the IFrame WebElement*/
WebElement iframeelement=driver.findElement(By.id("IF1"));

/*Switch to the IFrame By WebElement*/
driver.switchTo().frame("iframeelement");
```

WebElement is located by any of the location strategies and then passed to the switchTo() command

Few Additional Commands using Selenium WebDriver

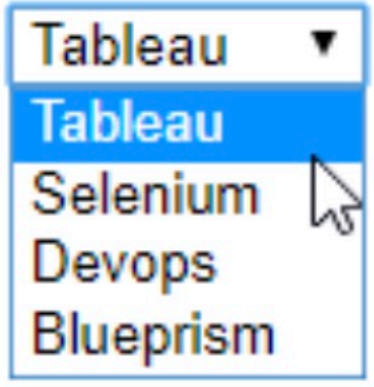
```
//Syntax to switch back to the Parent and Main Frames

/*Switch back to the Parent Frame*/
driver.switchTo().parentFrame();

/*Switch back to the Main Frame*/
driver.switchTo().defaultContent();
```

parentFrame() and defaultContent() methods are used to switch back to the Parent and Main Frames respectively

Testing a Dropdown

A Dropdown is a web element used to select a value from a series of options

- Test whether all the options are displayed in the dropdown or not
- Test whether an option is displayed at a specific position
- Test whether the selected value is same as that from the dropdown



What is a Select Class?

- Select Class is a WebDriver class that provides the implementation of the HTML <select> tag
- Initialize objects of Select type by passing the WebElement (Dropdown) as a parameter to its Constructor

```
/*Code snippet using Select Class*/

//Import the Select class
import org.openqa.selenium.support.ui.Select;

/*Locate the WebElement (Dropdown). The Dropdown element is declared as
*an instance of the Select Class
*/
WebElement testDropdwn = driver.findElement(By.id("SelectCourses"));
Select dropdwn = new Select(testDropdwn);

/*Alternatively, we can write the above statements as given below
*Select dropdwn = new Select(driver.findElement(By.id("SelectCourses")));
*/
```

Handling Dropdowns using Select Class Methods

SelectByIndex

Select an option based on index

SelectByValue

Select an option based on 'value' attribute

SelectByVisibleText

Select an option based on text displayed over the option

SelectByIndex() Method

SelectByIndex()

SelectByValue()

SelectByVisibleText

```
//Syntax for SelectByIndex() Method

Select oselect = new
Select(driver.findElement(By.id("SelectCourses")));

//Select the option at index 0
oselect.SelectByIndex(0);
```

SelectByIndex() method is used to select an option from the dropdown at index equal to 0

SelectByValue() Method

SelectByIndex()

SelectByValue()

SelectByVisibleText()

```
//Syntax for SelectByValue() Method

Select oselect = new
Select(driver.findElement(By.id("SelectCourses")));

//Select the option with value as "selenium"
oselect.SelectByValue("selenium");
```

SelectByValue() method is used to select an option with the value that matches the given argument

SelectByVisibleText()

SelectByIndex()

SelectByValue()

SelectByVisibleText()

```
//Syntax for SelectByVisibleText()Method

Select oselect = new
Select(driver.findElement(By.id("SelectCourses")));

//Select the option with text "Devops" over the option
oselect.SelectByVisibleText("Devops");
```

SelectByVisibleText() method is used to select an option from the dropdown displaying the text that matches with the given argument

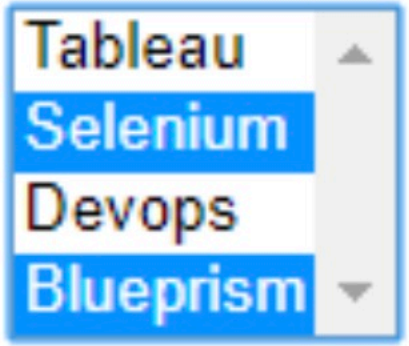
Importance of getOptions() Method in Select Class

```
//Syntax for getOptions() method

List<WebElement> elementCount=oSelect.getOptions();

System.out.println(elementCount.size());
```

getOptions() method is used to get all the options that belong to the <select> tag
It takes no parameters and returns List<WebElement>

How Multiple Select Method Works?

```
//Syntax for isMultiple() method

Select selectobj = new
Select(driver.findElement(By.id("SelectCourses")));

System.out.println(selectobj.isMultiple());

selectobj.selectByIndex(1);
selectobj.SelectByIndex(3);
```

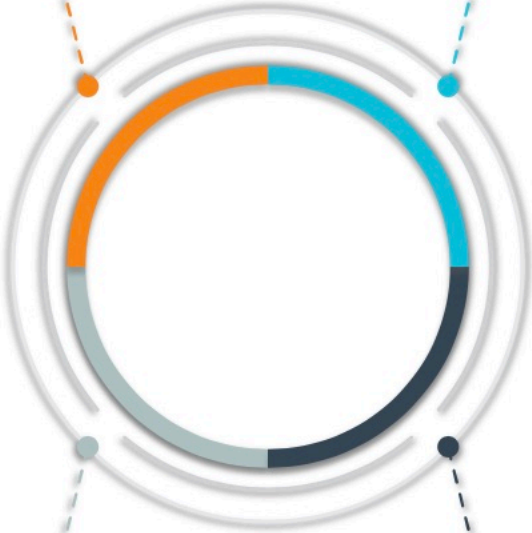


isMultiple() method tells whether or not the web element supports selection of multiple options at the same time

Handling Multi-Select Element using DeSelect Methods

deselectAll()
Clears all the selected options

deselectByValue()
Deselects the option whose “value” attribute matches the given parameter



deselectByIndex()
Deselects the option at the given index

deselectByVisibleText()
Deselects the option that displays the text matching the parameter

Actions Class & Action Interface

- Actions Class is an API used to emulate complex user interactions on the web
- Action is an Interface that is used to represent the single-user interaction to perform the series of actions build by Actions class

```
//Code snippet for Actions Class

/*Importing the packages*/
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.interactions.Action;

/*Instantiate the Actions class*/
Actions actions = new Actions(WebDriverObject);
```

Methods for Keyboard Based Interactions

keyDown()

sendKeys()

keyUp()

```
//API Syntax for 'keyDown()' method

public Actions keyDown(Keys thekey) throws
IllegalArgumentException

/* The IllegalArgumentException is thrown when the key
 * that is passed is not one amongst the Shift, Ctrl,
 * and Alt keys
 */
```

- **keyDown()** method is used for simulating the action of a key press
- Here, keys are referred to as Shift, Ctrl, and Alt keys

Methods for Keyboard Based Interactions

keyDown()

sendKeys()

keyUp()

```
//API Syntax for 'sendKeys()' method

public Actions sendKeys(CharSequence keysToSend)

/* This method sends keys in the form of sequence of
 * characters
 */
```

- **sendKeys()** method is used for sending keys to the active WebElement
- Enter alphanumeric and special character keys to Web Elements such as textbox or textarea

Methods for Keyboard Based Interactions

keyDown()

sendKeys()

keyUp()

```
//API Syntax for 'keyUp()' method

public Actions keyUp(Keys thekey)

/* keyUp() method releases the key that is pressed by
 * the keyDown() method
 */
```

- **keyUp()** method is used for simulating the action of a key release
- Perform keyUp() action only after the keyDown() action is being performed

Methods for Mouse Based Interactions

- **moveByOffset()**
- click()
- clickAndHold()
- release()
- moveToElement()
- dragAndDropBy()
- doubleClick()
- contextClick()

```
//API Syntax for 'moveByOffset()' method

public Actions moveByOffset(int xOffset, int yOffset)

/* xOffset and yOffset are the input parameters providing
 * the WebDriver the amount of offset to be moved along the
 * x and y axis
 */
```

- **moveByOffset()** method is used for moving the mouse cursor from its current position to another point on the web page

Methods for Mouse Based Interactions

- moveByOffset()
- **click()**
- clickAndHold()
- release()
- moveToElement()
- dragAndDropBy()
- doubleClick()
- contextClick()

```
//API Syntax for "click()" method

public Actions click()

/*Click at a current location action*/

public Actions click(WebElement OnElement)

/*Click on a WebElement action*/
```

- **click()** method is used for simulating the left-click of the mouse at its current location point
- **click(WebElement)** method is used to click directly on the specified WebElement

Methods for Mouse Based Interactions

- moveByOffset()
- click()
- **clickAndHold()**
- release()
- moveToElement()
- dragAndDropBy()
- doubleClick()
- contextClick()

```
//API Syntax for "clickAndHold()" method

public Actions clickAndHold()

/*Click and hold at current location action*/

public Actions clickAndHold(WebElement OnElement)

/*Click and hold on another WebElement action*/
```

- **clickAndHold()** method is used to left-click on an element and hold it without releasing it
- **clickAndHold(WebElement)** method is used to left-click on the specified WebElement and hold it without releasing the current position of the cursor

Methods for Mouse Based Interactions

- moveByOffset()
- click()
- clickAndHold()
- **release()**
- moveToElement()
- dragAndDropBy()
- doubleClick()
- contextClick()

```
//API Syntax for "release()" method

public Actions release()

/*Release on the current location action*/

public Actions release(WebElement OnElement)

/*Release on another WebElement action*/
```

- **release()** method is used to release the left-mouse button on a WebElement
- **release(WebElement)** method is used to release the currently held WebElement in the middle of another WebElement

Methods for Mouse Based Interactions

- moveByOffset()
- click()
- clickAndHold()
- release()
- **moveToElement()**
- dragAndDropBy()
- doubleClick()
- contextClick()

```
//API Syntax for 'moveToElement()' method

public Actions moveToElement(WebElement ToElement)

/* The input parameter is the target element where the
 * mouse cursor should be moved
 */
```

- **moveToElement()** method is used to move the mouse cursor to a WebElement on a web page

Methods for Mouse Based Interactions

- moveByOffset()
- click()
- clickAndHold()
- release()
- moveToElement()
- **dragAndDropBy()**
- doubleClick()
- contextClick()

```
//API Syntax for 'dragAndDropBy()' method

public Actions dragAndDropBy(WebElement source,int xOffset,
int yOffset)

/* The WebElement input parameter is the target
 * WebElement to be dragged. xOffset and yOffset is the
 * horizontal and vertical offset to be moved
 */
```

- **dragAndDropBy()** method is used to drag and drop the components or WebElements of a web page

Methods for Mouse Based Interactions

- moveByOffset()
- click()
- clickAndHold()
- release()
- moveToElement()
- **dragAndDropBy()**
- doubleClick()
- contextClick()

```
//API Syntax for 'dragAndDrop()' method

public Actions dragAndDrop(WebElement source,WebElement target)

/* WebElement source and WebElement target are the input
 * parameters of this method
 */
```

- **dragAndDrop()** method is used to move the WebElement on to the target element

Methods for Mouse Based Interactions

- moveByOffset()
- click()
- clickAndHold()
- release()
- moveToElement()
- dragAndDropBy()
- **doubleClick()**
- contextClick()

```
//API Syntax for 'doubleClick()' method

public Actions doubleClick()

/* Double click at current location action*/

 public Actions doubleClick(WebElement OnElement)

/* Double click on WebElement action*/
```

- **doubleClick()** method is used to perform double-click at the current mouse cursor location
- **doubleClick(WebElement)** method is used to emulate the double-click of a WebElement

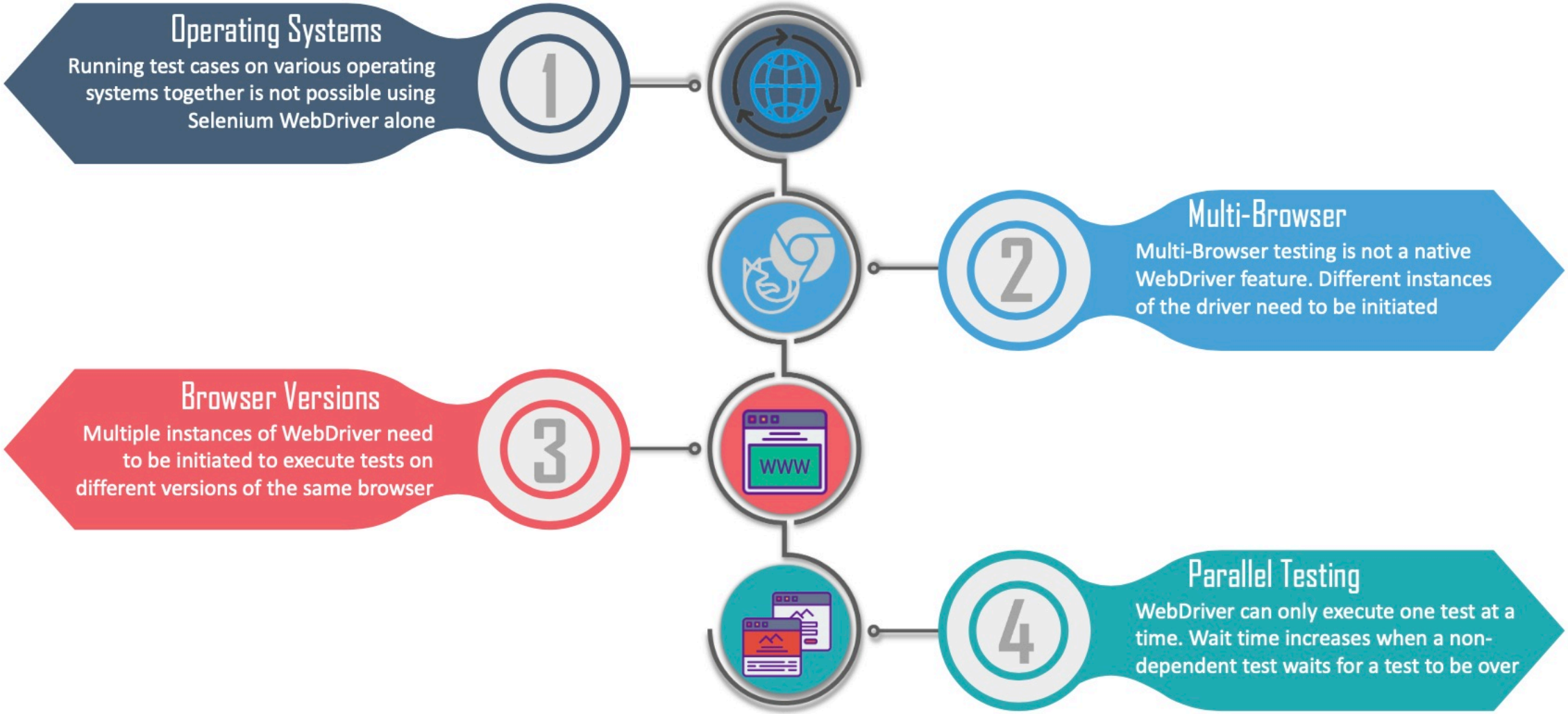
Methods for Mouse Based Interactions

- moveByOffset()
- click()
- clickAndHold()
- release()
- moveToElement()
- dragAndDropBy()
- doubleClick()
- **contextClick()**

```
//API Syntax for 'contextClick()' method

public Actions contextClick()

/* Context click at current location action*/

 public Actions contextClick(WebElement OnElement)

/* Context click on WebElement action*/
```

- **contextClick()** method is used to perform a context click at the current mouse location
- **contextClick(WebElement)** method is used to perform context click on a WebElement

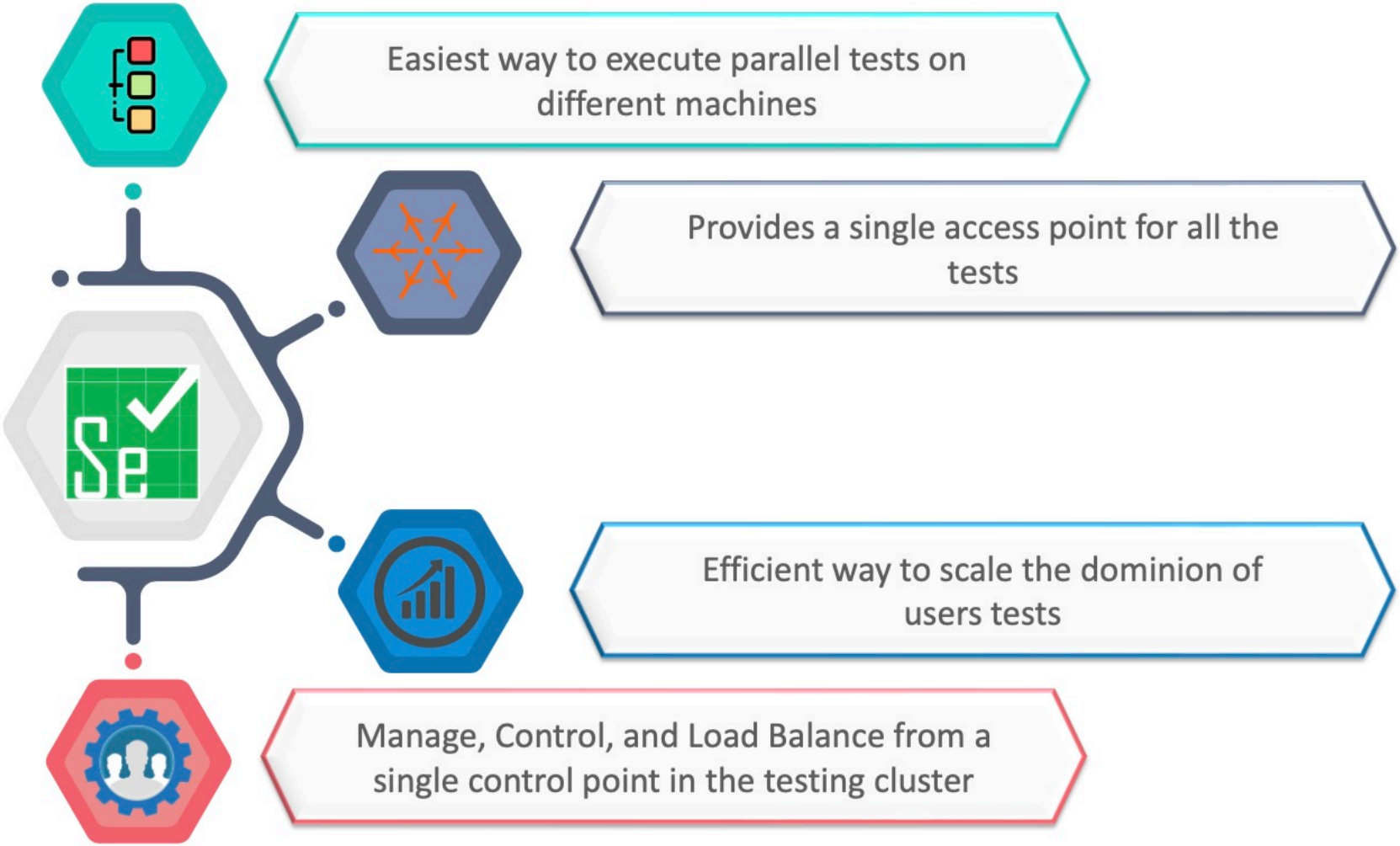
The Problem


Operating Systems
Running test cases on various operating systems together is not possible using Selenium WebDriver alone
1
Multi-Browser
Multi-Browser testing is not a native WebDriver feature. Different instances of the driver need to be initiated
2
Browser Versions
Multiple instances of WebDriver need to be initiated to execute tests on different versions of the same browser
3
www
Parallel Testing
WebDriver can only execute one test at a time. Wait time increases when a non-dependent test waits for a test to be over
4

The Solution: Grid

Selenium Grid acts as a proxy server which routes test commands to remote browser instances. The commands in JSON format are routed via the Grid server to the connected Grid nodes

Selenium Grid: Advantages



When to use Selenium Grid?

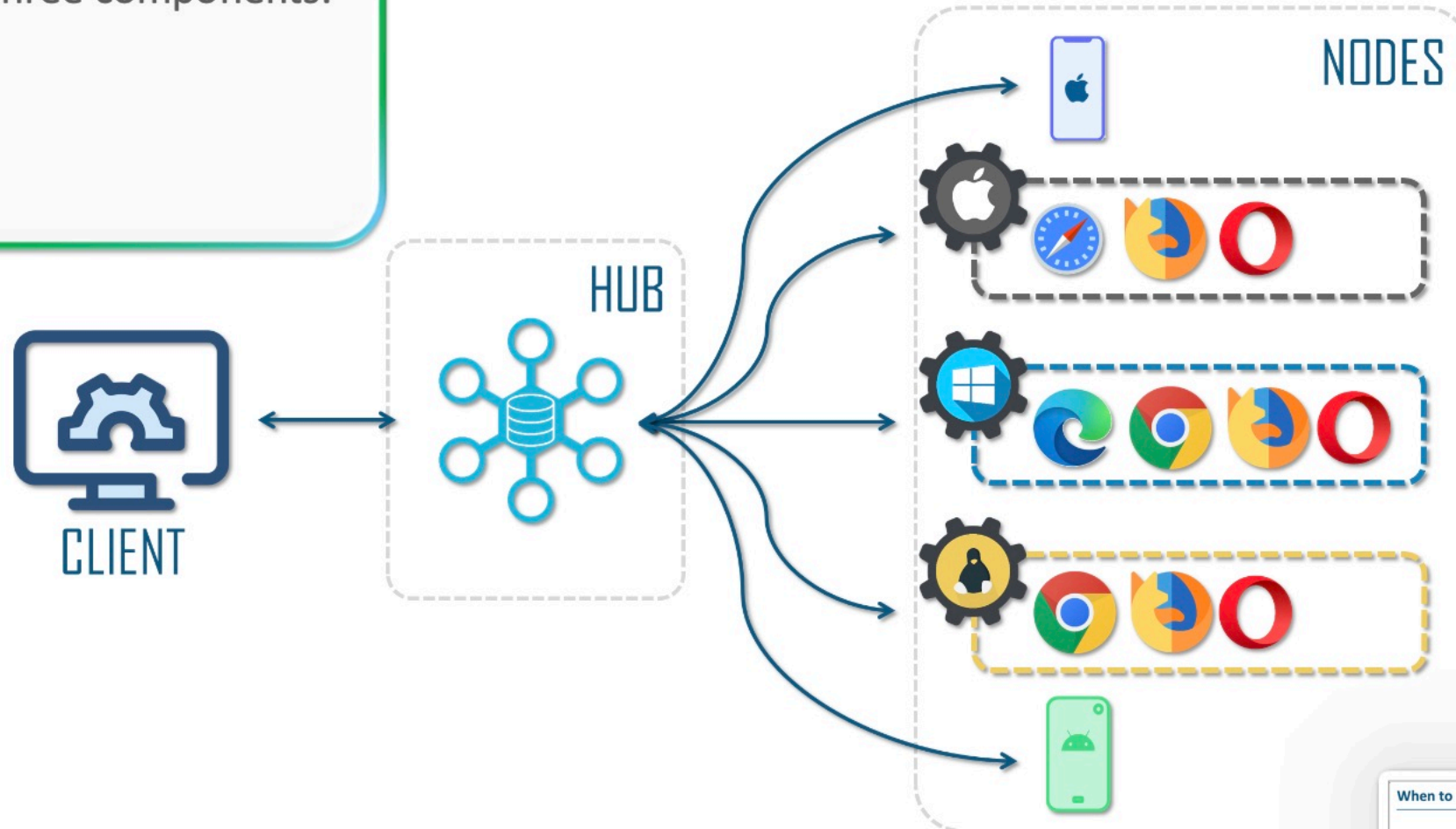
Selenium Grid is most prominently used in two scenarios:

- To execute tests on various platforms and browsers simultaneously
- To reduce overall time for the test suite to complete

Selenium Grid Architecture

Selenium Grid works using three components:

- Client
- The Hub
- The Nodes



Selenium Grid Architecture: Client

CLIENT

- The master machine which sends the test configuration to the Hub
- All the test scripts are written on the client
- The execution is initiated from the client
- Test results from all the nodes are reverted to the Client

CLIENT

Selenium Grid Architecture: Hub

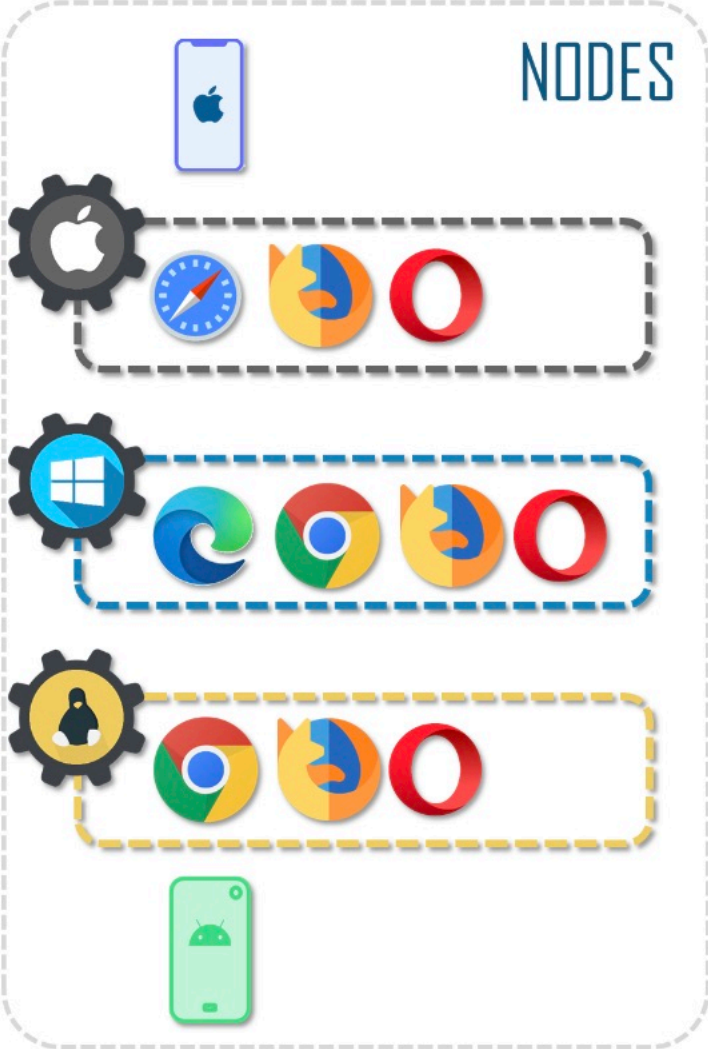
HUB

- Acts as the mediator between the client and the nodes
- Responsible for distributing and executing tests to the nodes
- Each Selenium Grid setup requires exactly one Hub
- Accepts instructions from the client and executes them on the remote nodes
- Also known as Grid Server

Selenium Grid Architecture: Nodes

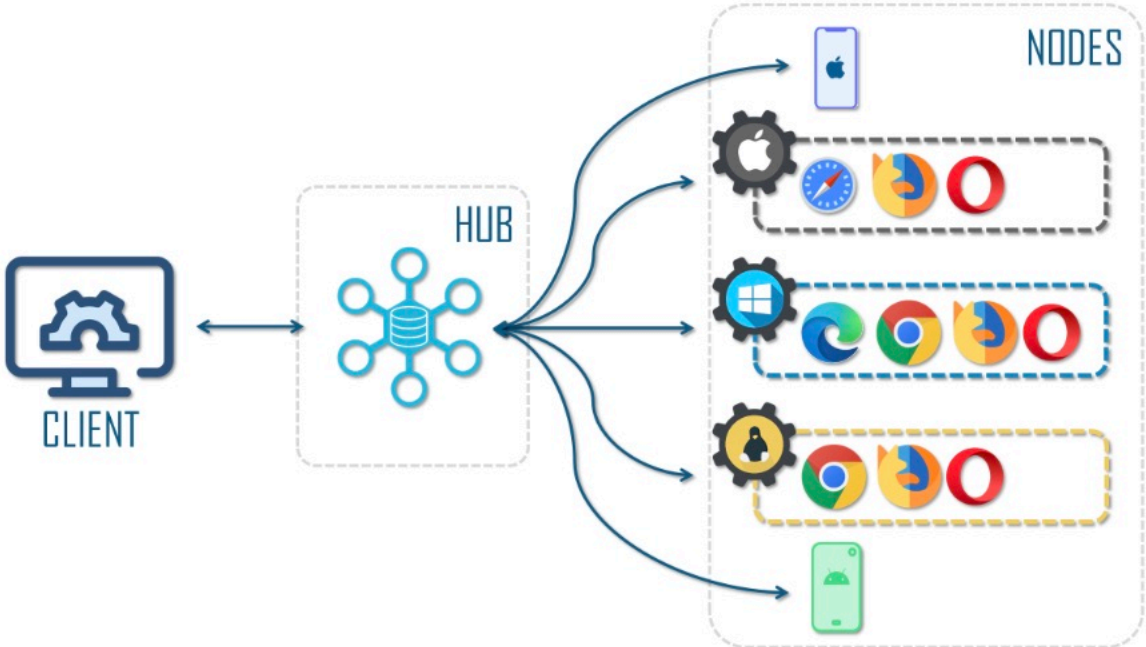
NODES

- All the browsers for testing reside on the connected nodes
- Each nodes is registered with the Hub
- Nodes share their capabilities to the Hub
- Nodes can be of various platforms and can run different browsers



Selenium Grid: Working

- Grid works as a network of one hub and the attached nodes
- The hub is aware of the resources, configuration and the browsers present on the node
- Each node has its own instance of webdriver
- The hub reads the desired capabilities object and redirects requests to a matching node



Selenium RemoteWebDriver

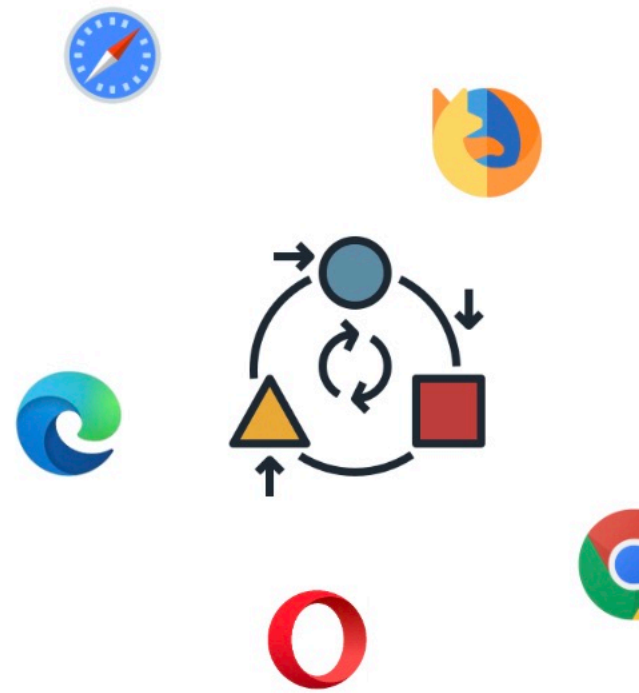
Selenium Grid uses Selenium RemoteWebDriver to manipulate the browsers in the node machines attached to the hub

Selenium RemoteWebDriver

1. The RemoteWebDriver sends request for controlling the web browser to a web server
2. Selenium server waits for the RemoteWebDriver Connection
3. There is no visual feedback for web browser like with the normal WebDriver
4. Useful for managing browsers on remote machines and to run tests asynchronously

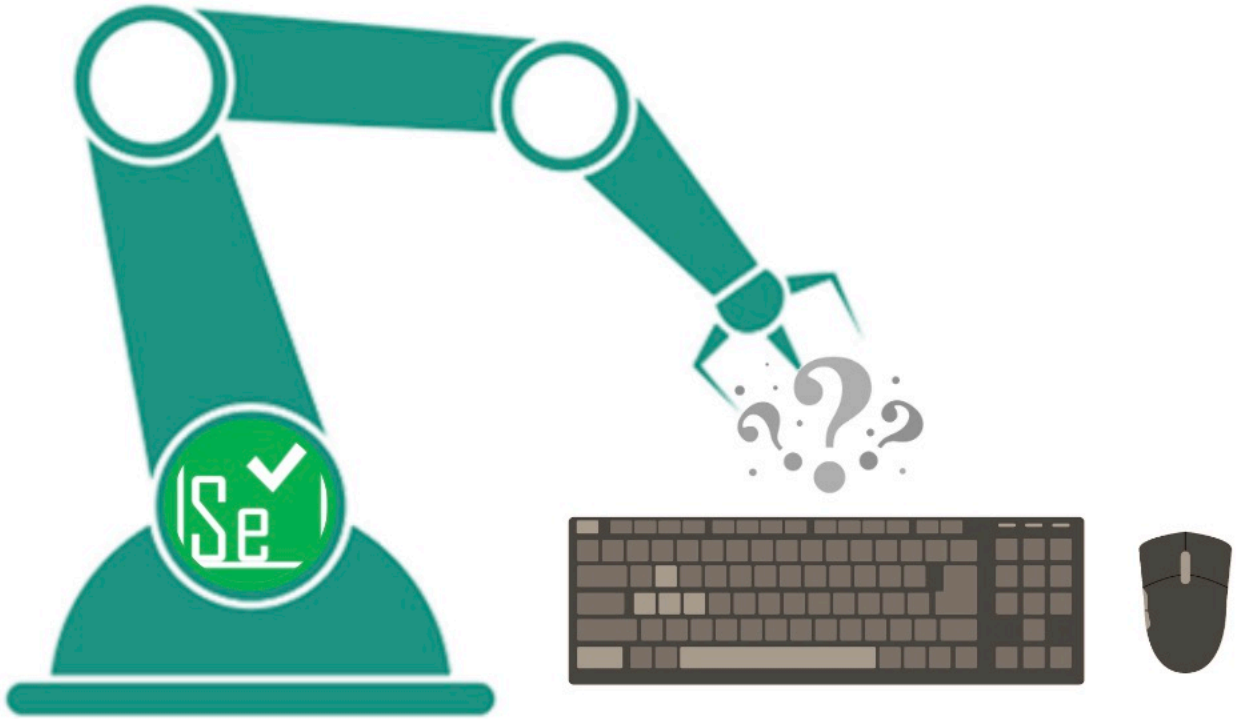
Importance of Desired Capabilities in Grid

- Defines a set of key-value pairs for the environment where the tests are executed
- Grid uses Desired Capabilities to define individual aspects of each browser it runs the tests on
- Properties such as PlatformName, Version of browser, BrowserName, etc. are defined using this class
- Uses the setCapability() method to define the key-value pairs



The Problem

- Selenium does not have the provision to automate keyboard and mouse gestures on its own
- Automation tests usually fail if pop-ups from the native OS application come up



Robot Class

Robot class enables selenium users to make use of keyboard and mouse events in their automation scripts

Advantages of Robot Class

Enables control over keyboard and mouse events

Useful in handling upload/download actions

Provides ability to manage OS level pop-ups

Easily integrated with java selenium scripts as it is a part of java package

Limitations of Robot Class

Dependent on Screen Resolution of the test window

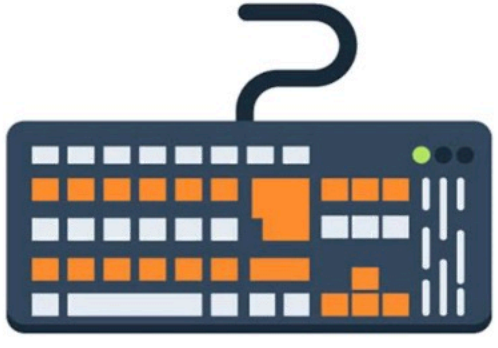
Difficulty in managing multiple windows together

No defined methods to switch between frames

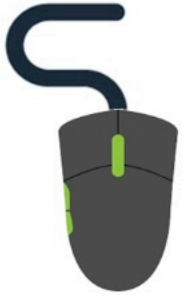
Robot Class Methods

Robot Class has:

- Two methods for Keyboard Events
- Three methods for Mouse Events



- keyPress()
- keyRelease()



- mouseMove()
- mousePress()
- mouseRelease()

Robot Class Methods

Keyboard Methods

- keyPress()
- keyRelease()

Mouse Methods

- mouseMove()
- mousePress()
- mouseRelease()

- The keyPress() is used to simulate a key press on the keyboard

```
Syntax: keyPress(KeyEvent.keyName)
```

- The keyRelease() is used to release a key pressed using the keypress()

```
Syntax: keyRelease(KeyEvent.keyName)
```

Robot Class Methods

Keyboard Methods

keyPress()

keyRelease()

Mouse Methods

mouseMove()

mousePress()

mouseRelease()

- The mouseMove() moves the cursor to the given X and Y co-ordinates

```
Syntax: mouseMove(x co-ordinate, y co-ordinate)
```

- The mousePress() is used to press one or more mouse buttons

```
Syntax: mousePress(InputEvent.buttonName)
```

- mouseRelease() releases an already pressed mouse button

```
Syntax: mouseRelease(InputEvent.buttonName)
```

What is Selenium IDE?

- Selenium IDE is an Open Source Record and Playback Tool used for Automation Testing

- Selenium IDE allows to record user interactions with websites
- Easy-to-use Firefox and Chrome extension
- Records and plays back tests on Firefox and Chrome
- Easy-to-manage since tests are organized into suites
- Saves and loads scripts for future playbacks

Advantages of Selenium IDE

Saves Time

Performs quick authoring of reliable end-to-end tests thereby saving a lot of time

Easy to Debug Tests

Features such as setting breakpoints and pausing on exceptions are present thereby making it easier to debug tests

Cross-Browser Execution

Using Command-line Runner, tests could run on any browser/operating system combination

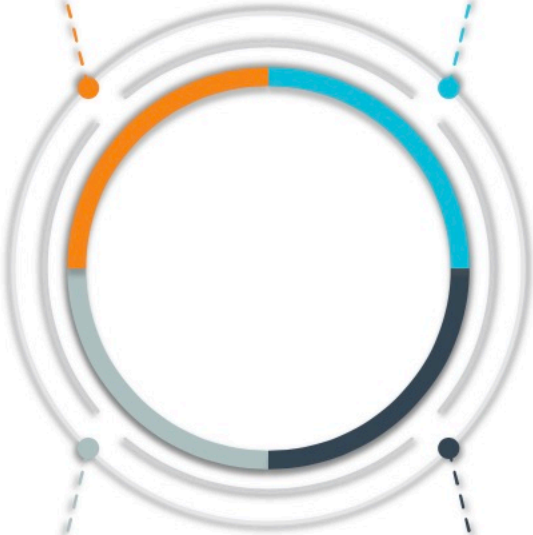
Features of Selenium IDE

Resilient Tests

Selenium IDE records multiple locators for each web element it interacts with. If one of the locators fail, the remaining ones are tried

Control Flow

A control flow structure is available with commands such as if, while and times



Test Case Reuse

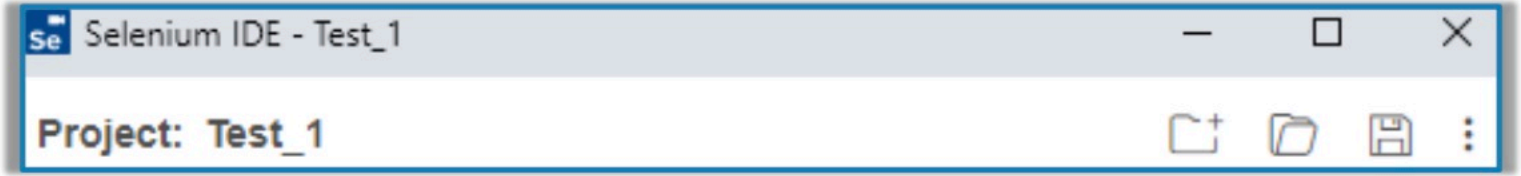
Through run command, a test case can be reused within another that allows a logic to be reused at multiple places

Extendable through Plugins

IDE could be extended through plugins thereby introducing new commands or integrating with a third party service

Selenium IDE Interface

1. Menu Bar
2. Tool Bar
3. Test Case Pane
4. Output Pane



Icon	Name	Description
	Create new project	Create a new project
	Open project	Open an existing project
	Save project	Save a project

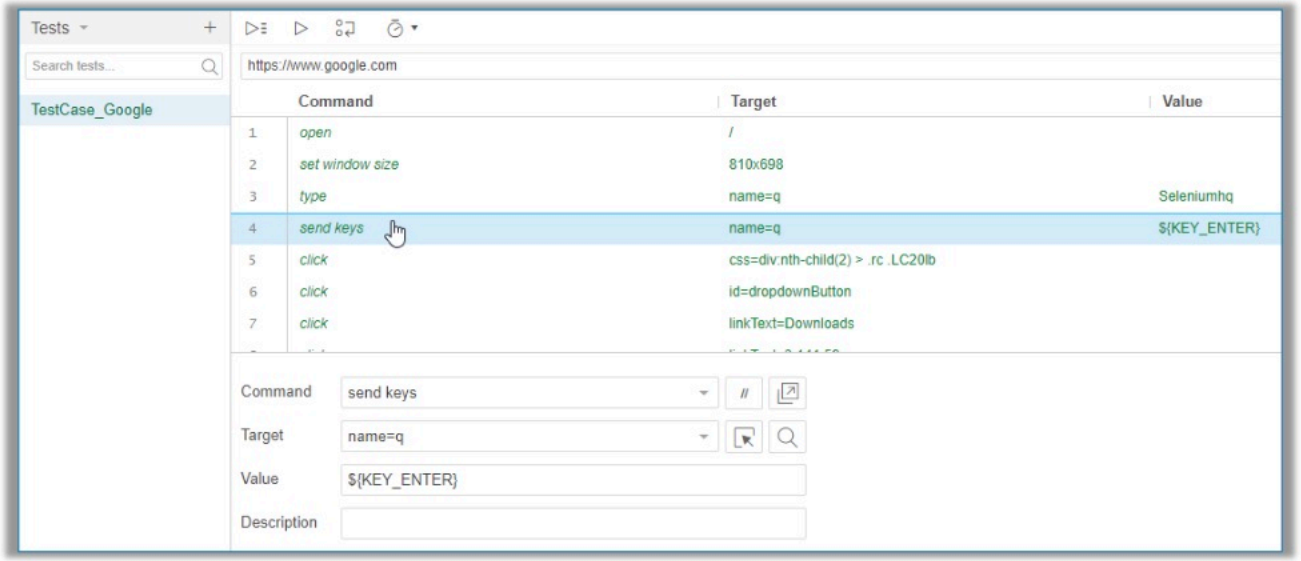
Selenium IDE Interface

1. **Menu Bar**
2. **Tool Bar**
3. **Test Case Pane**
4. **Output Pane**

Icon	Name	Description
	Run all tests	Runs the test suite wherein multiple tests are loaded
	Run current test	Runs the currently selected test
	Step over current command	Allows to “step” through a test case, runs one command at a time
	Test execution speed	Setting up the speed of the test
	Disable breakpoints	Disables the breakpoint options
	Pause on exceptions	Pauses execution of a test if an exception occurs
	Start recording	Records user’s interactions on the web page via browser

Selenium IDE Interface

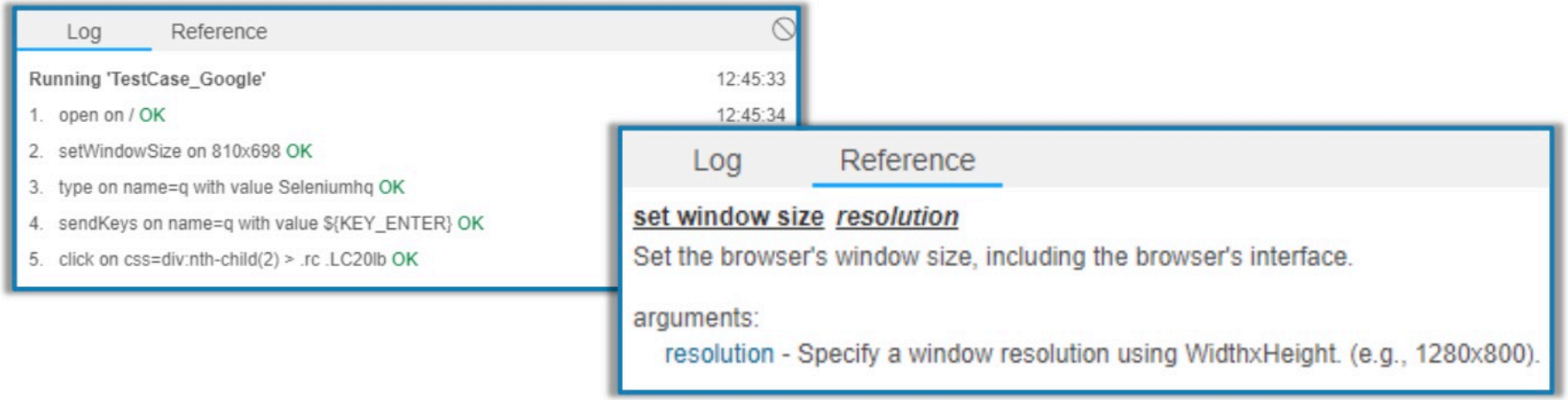
1. Menu Bar
2. Tool Bar
3. Test Case Pane
4. Output Pane



Name	Description
Command	Field having all the recorded commands
Target	Field having all the recorded web elements id
Value	Value stored on the recorded web element

Selenium IDE Interface

1. **Menu Bar**
2. **Tool Bar**
3. **Test Case Pane**
4. **Output Pane**

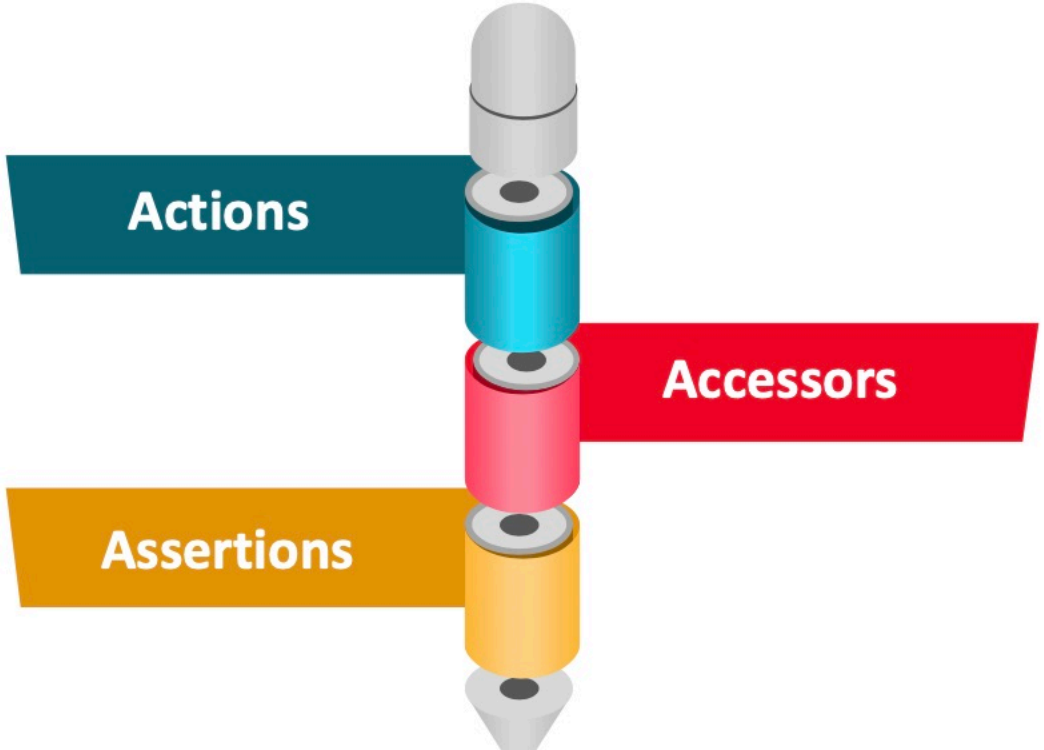


Name	Description
Log	Displays error messages and command information
Reference	Provides documentation of selected commands

What is Selenese?

- Selenese is the set of Selenium commands used to test web applications
- In a test script, these commands are executed in a sequence

Types of Selenium Commands



Types of Selenium Commands

1. **Actions**
2. **Accessors**
3. **Assertions**

- Actions are commands that directly interacts with web elements
- Actions are used for simulating user's actions when interacting with web elements
- Test execution stops if an action fails or has an error
- For example:
  - "click" command is an action since there is a direct interaction with the web element clicked
  - "type" command is an action since there is a direct interaction with the web element wherein the values are entered into it

Types of Selenium Commands

1. **Actions**
2. **Accessors**
3. **Assertions**

- Accessors are commands that check the state of a web element and stores the application state in a user-defined variable
- Accessors are used to evaluate whether the desired result is present on the web page and stores the result in a user-defined variable
- For Example:
  - storeTitle - Reads the web page title and saves it in a user-defined variable. It does not interact with any web element on the page
  - storeTextPresent - If the desired text is found on the web page, it stores boolean value as true else stores false
  - storeElementPresent - If the desired web element is present on the web page, it stores boolean value as true else stores false
  - storeAllLinks - Reads and stores all the hyperlinks available within the web page into a user-defined variable

Types of Selenium Commands

1. **Actions**
2. **Accessors**
3. **Assertions**

- Assertions verifies whether the state of a web element conforms to the expected state
- In Test Automation, Assertions are used as verification or checkpoints
- Forms of Assertions:
  - assert : This command ensures that the test execution is terminated in case of a failure
  - verify : This command lets the Selenium IDE continue with the test execution even if the verification fails
  - waitFor : This command waits for a certain condition to be met before executing the next test step

Commonly used Selenium IDE Commands

Command	Target	Value	Comment
click, clickAndWait	A Locator	-	Clicks on an element. The …AndWait version waits for a Page Load Event
open	A URL	-	open supports full URLs such as https://abc.de. Base URL is ignored
pause		Amount of sleep time in milliseconds	For Example: “5000” means sleep time of 5 seconds
select, selectAndWait	A Locator of a drop-down menu	An Option Locator	For Example: ”label=Option1”. The …AndWait waits for a Page Load Event
selectFrame	“index=0” , “relative=parent” “relative=top”	-	Works for Frames and IFrames
sendKeys	A Locator	A Character	For Example: Value=“${KEY_ENTER}”
type	A Locator	The string to be set to an input field	For Example: Value=“Seleniumhq”

Commonly used Selenium IDE Commands (Contd…)

Command	Description
assertTitle, verifyTitle	▪ Returns the current page title and compares it with the specified title
assertTextPresent, verifyTextPresent	▪ Asserts or verifies the presence of a text within the web page
waitForElementPresent	▪ Pauses the test execution and waits until the specified element is present on the web page
waitForPageToLoad	▪ Pauses the test execution and waits until the web page is loaded completely

Building a Test Case in Selenium IDE

Recording

- Record a test-case by interacting with the website
- Commands are inserted automatically based on the actions
- Typically, includes:
  - Click or ClickAndWait
  - Type command
  - Select command

Verification And Assertions

- Check the properties of a web-page
- Use assert and verify commands

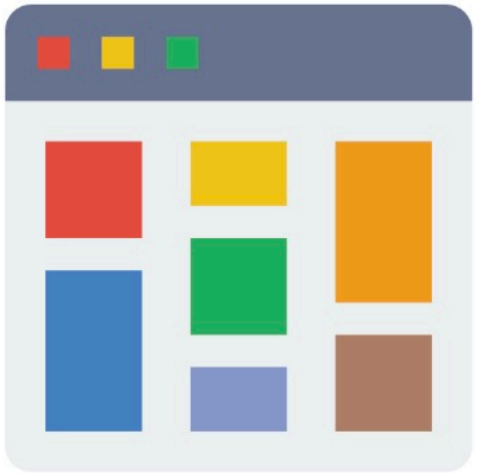
Editing

- Insert Command
- Insert Comment
- Edit a Command or Comment

Why Use Page Object Model?

Problem Statement:

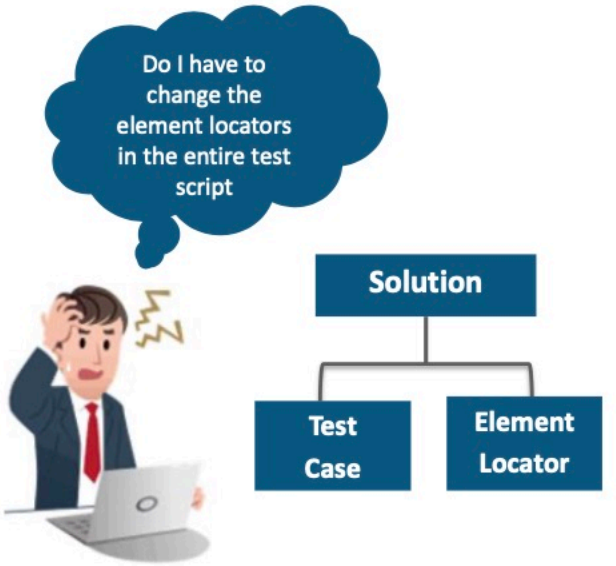
- Consider a web page having 'n' number of elements, which are used in multiple test cases
- What will happen if a path/locator of a web element changes?



A web page with 'n' elements

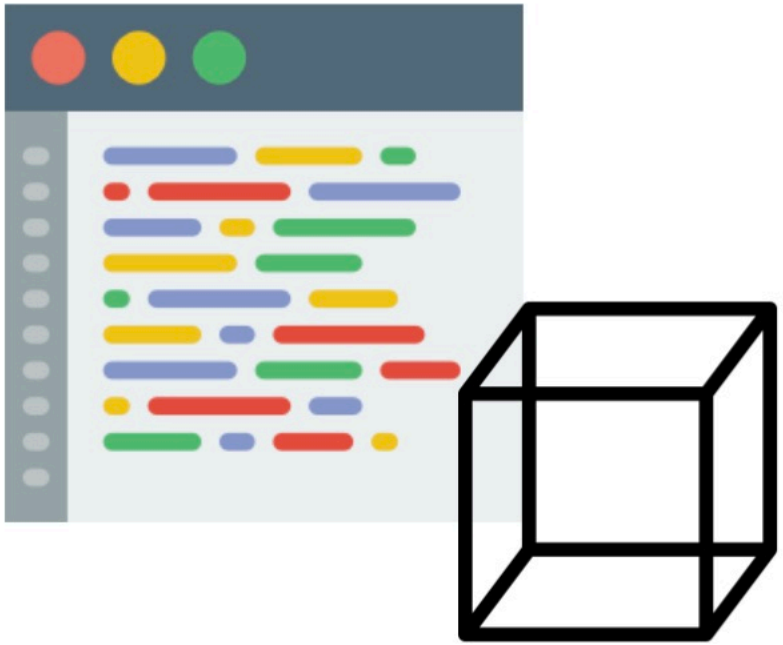
```
driver.get(http://the-internet.herokuapp.com);

driver.findElement(By.
cssSelector("a:contains("Forgot
Password")")).click();

driver.findElement(By. cssSelector("input#email
")).sendKeys("abc@gmail.com");

driver.findElement(By. cssSelector
("button.radius")).click();
```

Test Script



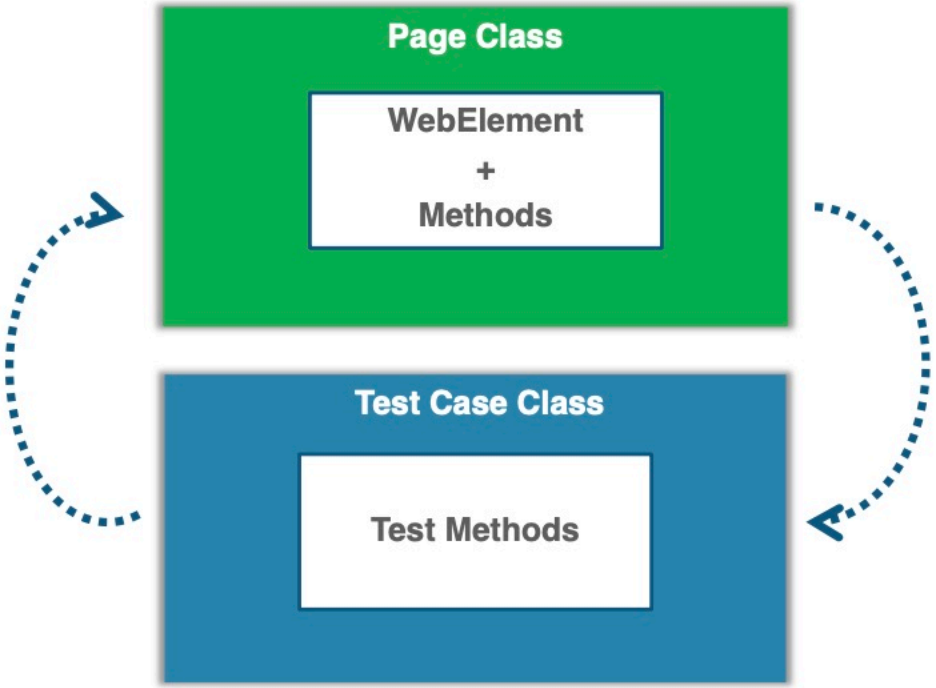
What is a Page Object?

- Page Object is an object-oriented class that serves as an interface to the page of a web application under test
- Tests use the methods of this Page Object class for interacting with the User Interface of the web page
- If there is change in the User Interface of a web page, tests need not be changed, only the code within the page object needs to be changed



What is Page Object Model?

- Page Object Model is a design pattern for creating Object Repository for web elements
- Here, all the element locators used by the web application are stored in a class file separately from the test case



POM Design Pattern

Page Class

It contains web element locators and methods for performing operations on the web elements

Test Case

It contains test scenarios and calls the methods present in the Page Class

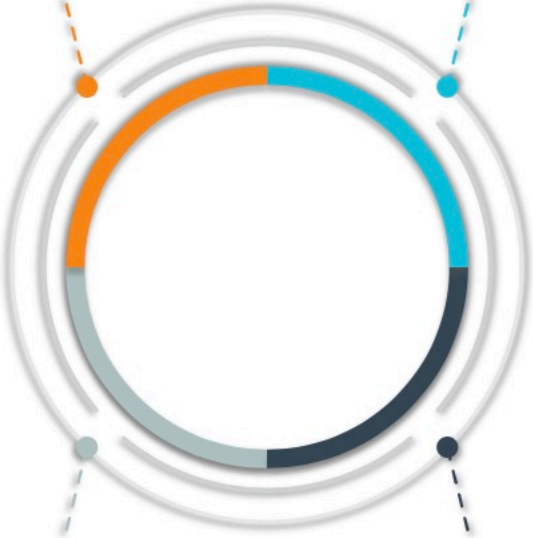
Advantages of Page Object Model

Code Reusability

Code Reusability is achieved by writing the code once and reusing it in difference tests

Object Repository

Page Object Repository is independent of automation tests which could be reused for different purpose with different tools



Code Maintainability

Operations and flows in the UI are kept separately from verification that keeps the code clean, easy to understand and maintain

Readability

Improves readability since the test code is separated from the page specific code

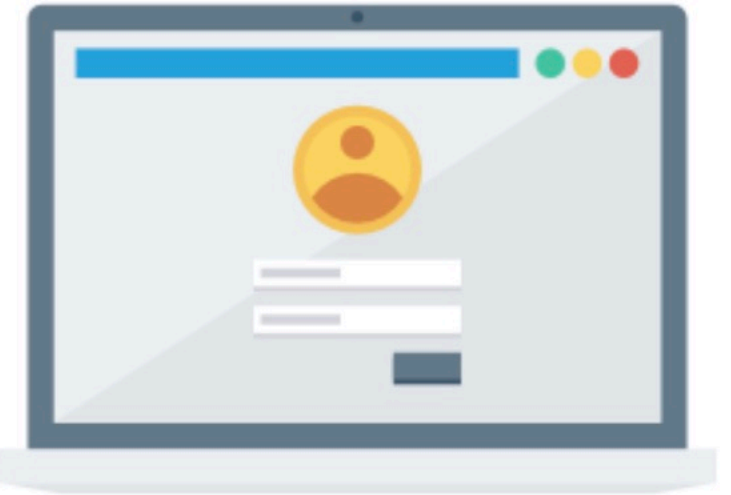
What is Page Factory?

- Page Factory is an extension to the Page Object Model
- It is used to initialize web elements that are defined in Page Objects

- With Page Factory,
  - **@FindBy** annotation is used to identify the web elements in a web page
  - **initElements()** method is used to initialize the web elements
  - **@CacheLookUp** annotation is used to store web page elements in cache memory

Implementing POM with Page Factory

- Consider a Login Page with username, password and a login button
- Following are few points to consider for implementing a POM with Page Factory for the above scenario



1. Understand the situation where we can use Page Factory
2. Implement Page Factory in the Login test case
3. Use Annotations of Page Factory
4. Divide the single Page Factory objects into different page classes
5. Combine the set of repetitive actions into methods

What is a Test Automation Framework?

Testing Frameworks is a set of protocols around which the tests cases are created and designed. It combines best practices and tools together for QA Professionals to test efficiently

Advantages of using a Test Automation Framework

Better Test Efficiency
Utilises the testing resources more effectively

Reduced Maintenance Costs
Properly written and maintained tests reduce overall testing costs

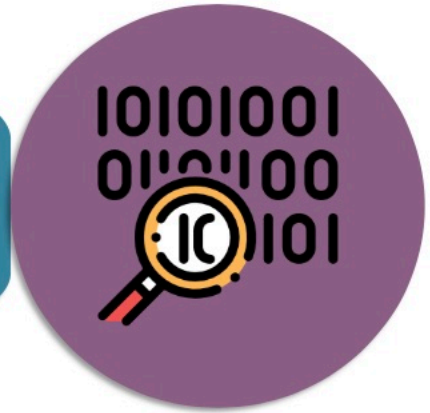
Reduced Manual Efforts
Manual intervention required is reduced substantially

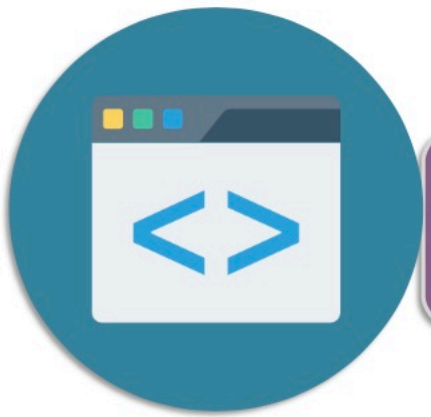
Improved Test Coverage
Tests written cover almost all of the source code of the application

Code Script Reusability
Enables reusing code for similar test cases

What is Selenium Framework?

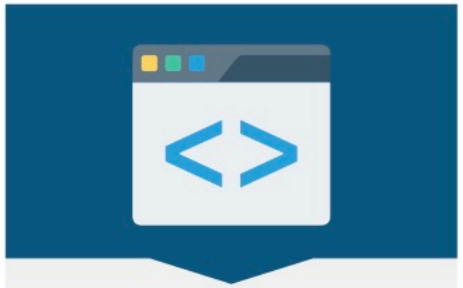
Selenium offers flexibility to create a testing framework, that can be reused



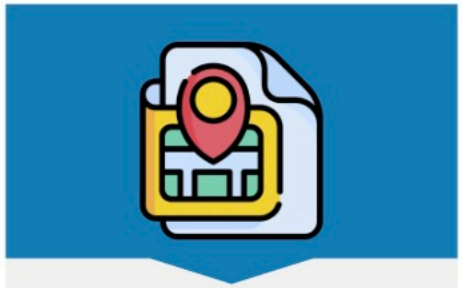


Selenium Framework is a code structure, that helps to make code maintenance easy

Benefits of Selenium Frameworks



Increased code re-usage



Higher portability

Reduced script maintenance cost

Higher code readability

Types of Automation Frameworks

The four most widely used Automation Frameworks in the industry are:

Data-Driven Framework

Keyword-Driven Framework

Hybrid Testing Framework

BDD Framework

Types of Automation Frameworks

Data-Driven Framework

Helps separate the script logic and the test data when similar tests are required to run on different data sets

Keyword-Driven Framework

Utilises independent data table of keywords which lays out step-by-step instruction for each test

Hybrid Testing Framework

A combination of two or more frameworks, integrating the strengths of each while removing the shortcomings

Behaviour Driven Development(BDD)

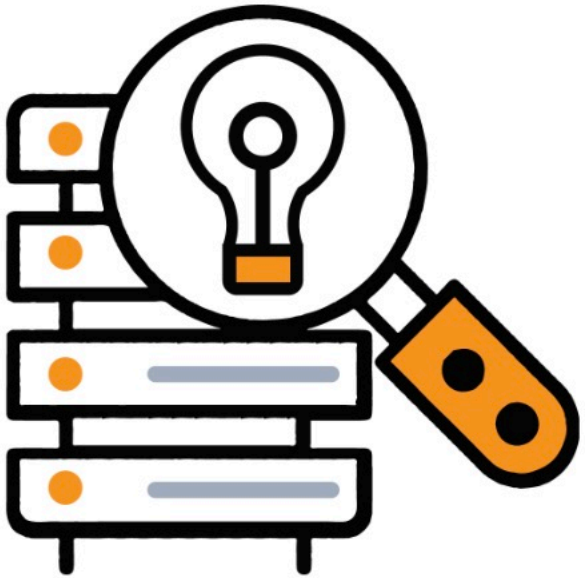
Creates test-cases in simple language which can help the non-tech audiences to understand what is happening in the project

Why use Data-Driven Testing Framework?

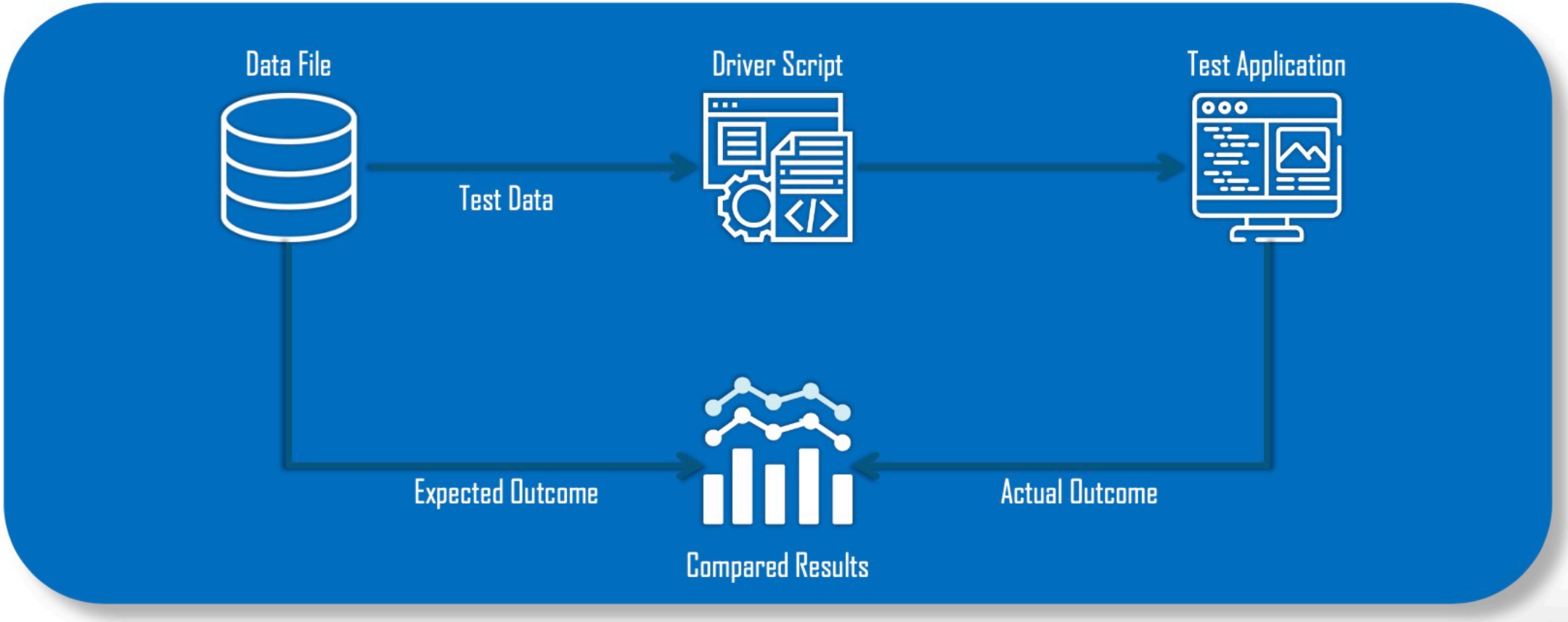
- Some applications require testers to execute similar tests but with different data sets
- For such cases, hard-coding the test data becomes inefficient

What is Data-Driven Testing Framework?

- Data Driven framework separates the test logic and the test data
- The data can be stored in tabular format or in spreadsheets
- Input values for the tests are stored in data files which can be in xls, xml, csv, etc. format
- Data is fed through variables inside the script



Data-Driven Testing Framework



Advantages of Data-Driven Testing Framework

Advantages

Same test can be performed on various data sets

Test coverage increases by varying the test data, also reducing the number of scripts needed

Avoids hard-coding the data inside of the script making the code independent of the test data

Number of test-cases executed in a shorter period of time increases substantially

Limitations of Data-Driven Testing Framework

Limitations

The initial setup of the framework takes a significant amount of time

Quality of the tests is entirely dependent on the competence of the team

An experienced tester is required to fully utilize the framework design

Significant amount of documentation is required to manage scripts, infrastructure, etc

Apache POI(Poorly Obfuscation Implementation)

- Selenium WebDriver does not have the provision to handle data from Microsoft Excel
- Apache POI is a Java API which enables read and write operations on excel files
- It enhances the test coverage and reduces maintenance

Apache POI: Interfaces

Interface	Description
XSSFWorkbook	Interface used to represent a workbook in a XLSX file
HSSFWorkbook	Interface used to represent a workbook in a XLS file
XSSFSheet	Interface used to represent a sheet in a XLSX file
HSSFSheet	Interface used to represent a sheet in a XLS file
XSSFRow	Interface used to represent a row in a selected sheet of XLSX file
HSSFRow	Interface used to represent a row in a selected sheet of XLS file
XSSFCell	Interface used to represent a cell in a selected row of XLSX file
HSSFCell	Interface used to represent a cell in a selected row of XLS file

Apache POI: Cell Field Options

Interface	Description
CELL_TYPE_BLANK	Represents a blank cell in a table
CELL_TYPE_BOOLEAN	Represents a Boolean value (True/False) in a cell
CELL_TYPE_ERROR	Represents an error value in a cell
CELL_TYPE_FORMULA	Gives a formula result on a cell
CELL_TYPE_NUMERIC	Represents a numeral value in a cell
CELL_TYPE_STRING	Represents a text string value in a cell

DataProviders

- The DataProvider Annotation enables the users to utilise various data sets in a single test suite
- It streamlines the way the data from different data providers can be fetched

DataProviders

1. The DataProvider annotation is a part of the TestNG framework
2. A method with a @DataProvider annotation is initialized where data fetching logic is written
3. The DataProvider method returns a 2D array which provides the test method with the required data
4. The DataProvider name is specified in the @Test annotation

Data-Driven Testing Best Practices

Data Separation

Avoid hard-coding test-data into the test-script. Separate the test-data and the test-script wherever possible

Data Coverage using Real Data

Positive/negative test-data should be used to cover success/failed test cases. Try ensuring test-data is close to the real data for which application was purposed

Clean-up Module

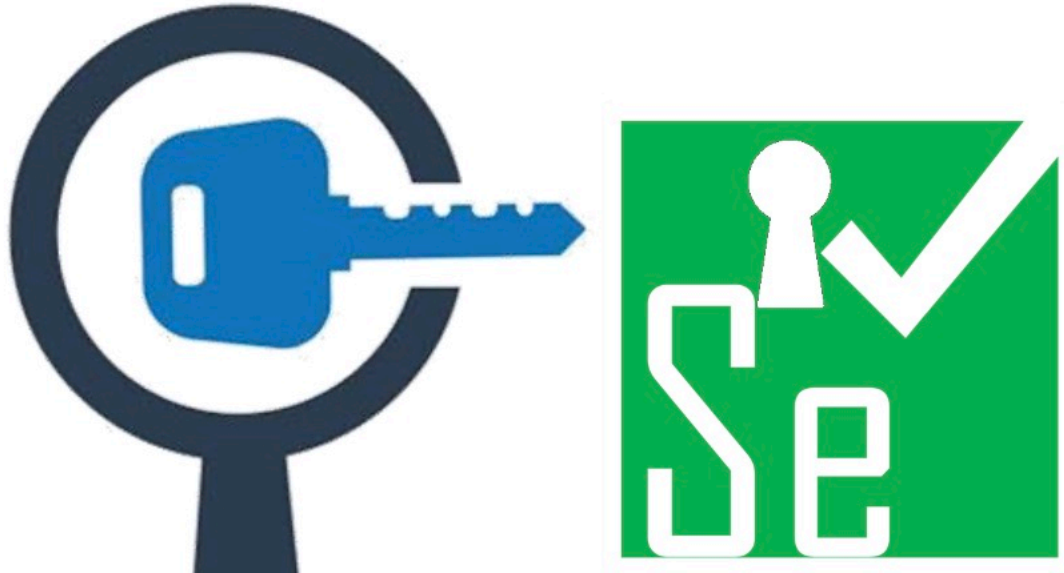
Ensure after every test run, a clean-up module should be executed which removes any changes made in the previous run

Error Handling

Error-handling logic should be inserted for all the negative test cases. This may let the test to continue or move on to the next test case

Keyword-Driven Testing Framework

Keyword-Driven Framework requires creating a separate table with a series of instructions for the tests which are executed



Why use Keyword-Driven Testing Framework?

- Keyword Driven framework separates the test cases and test steps from the coding
- Enables non-tech people to understand the automation process
- Enables manual testers to write automation scripts

Advantages of Keyword-Driven Testing Framework

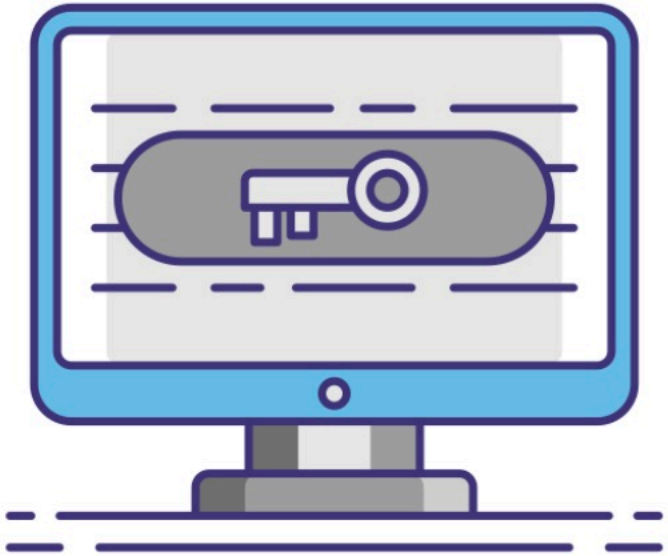
Advantages

Test plan can be laid out in simple language in the form of spreadsheets

After the initial setup, manual testers/non-tech testers can write the automation scripts

The object repository for tests can be created during the application development

Modular nature enables reusability of entire components and code as well



Limitations of Keyword-Driven Testing Framework

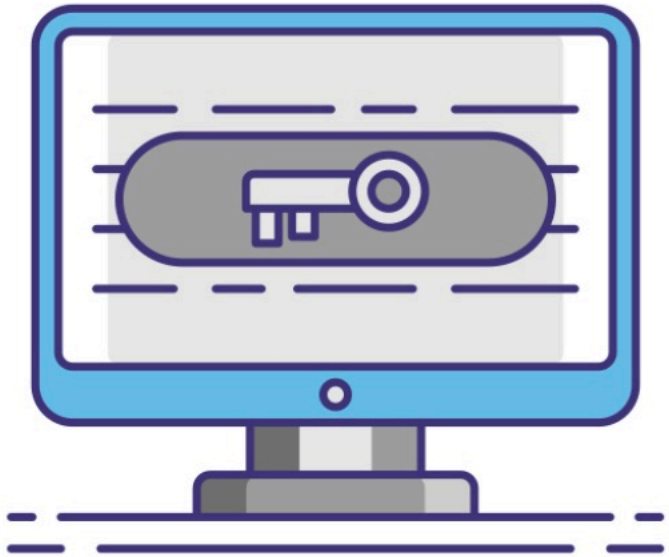
Limitations

Initial cost of framework setup is high and laborious

Defining keywords and object repository becomes tedious

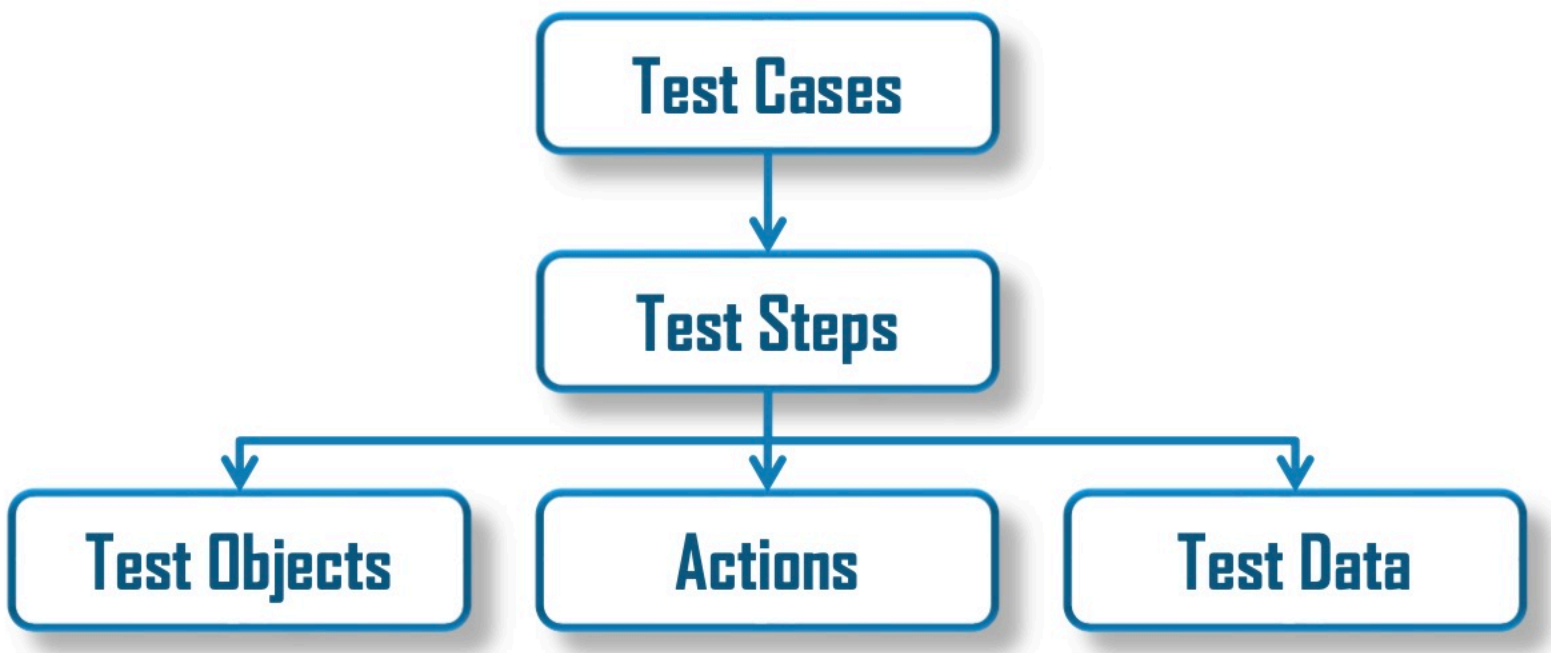
To maintain the test standards, a well rounded automation engineer is required

Maintaining keywords while scaling test operations is difficult

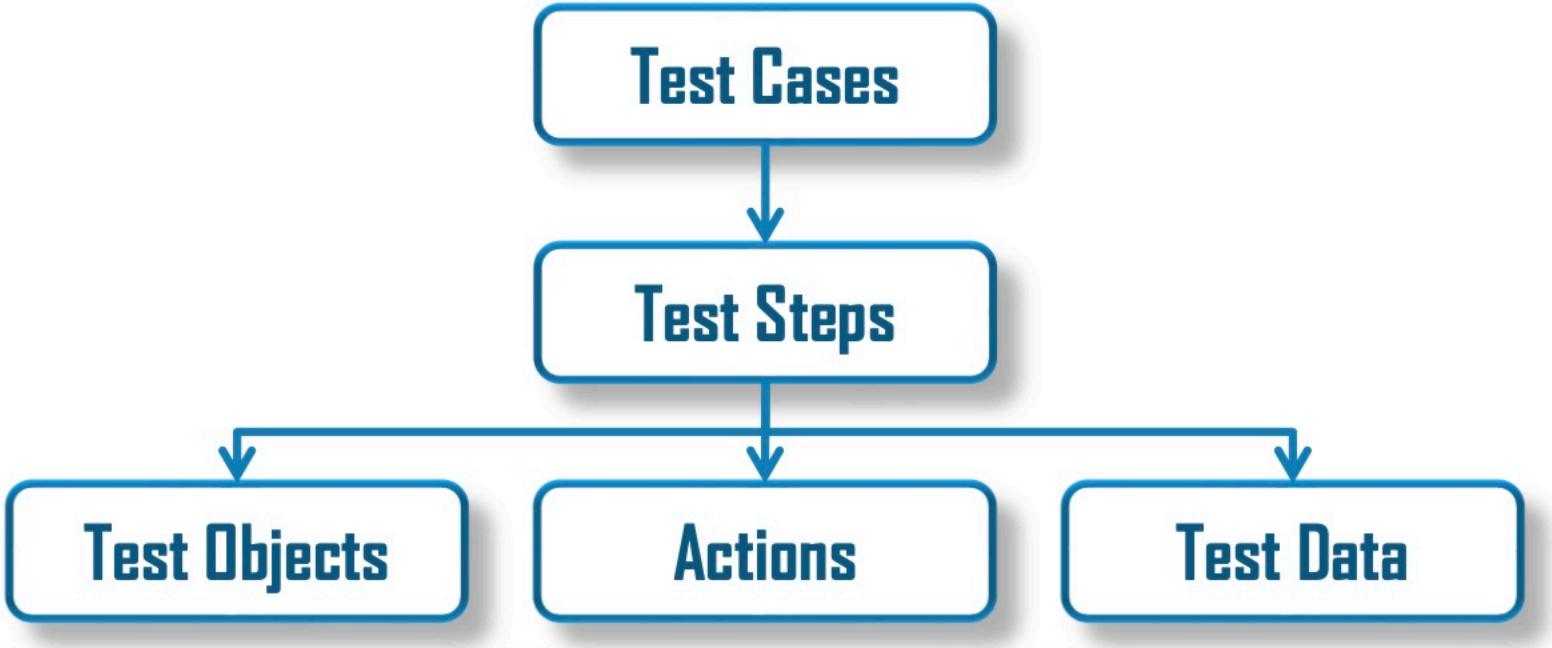


Keyword-Driven Framework Working

Keyword Driven Framework requires the test-cases to be divided into modular pieces. This increases reusability and modularity of test-cases



Keyword-Driven Framework Working



Test Cases:-

- Test Steps : Description of the test being performed
- Test Objects : Name of the web element the action is performed on
- Actions : Name of the action being performed such as click, textInput, etc.
- Test Data : Required data values used to perform the actions

Sample Keyword-Driven Framework Excel Sheet

Test Step	Keyword	Web Element	Data
Start Browser	launch	na	Chrome
Open Youtube	websiteSelect	na	https://www.youtube.com/
Search recipe	searchEdit	ytd_searchbox	Hawaiian Pizza
Select the result	videoSelect	ytd-video-renderer	3
Show Descritpion	showDesc	more	

Keyword-Driven Testing Framework Components

Excel Sheets

The sheet to keep track of data like the test cases, test steps, Objects and actions

Data Sheet

Excel File that stores the data values for the Test Objects

Object Repository

Web elements property store which are linked to the objects in the tests

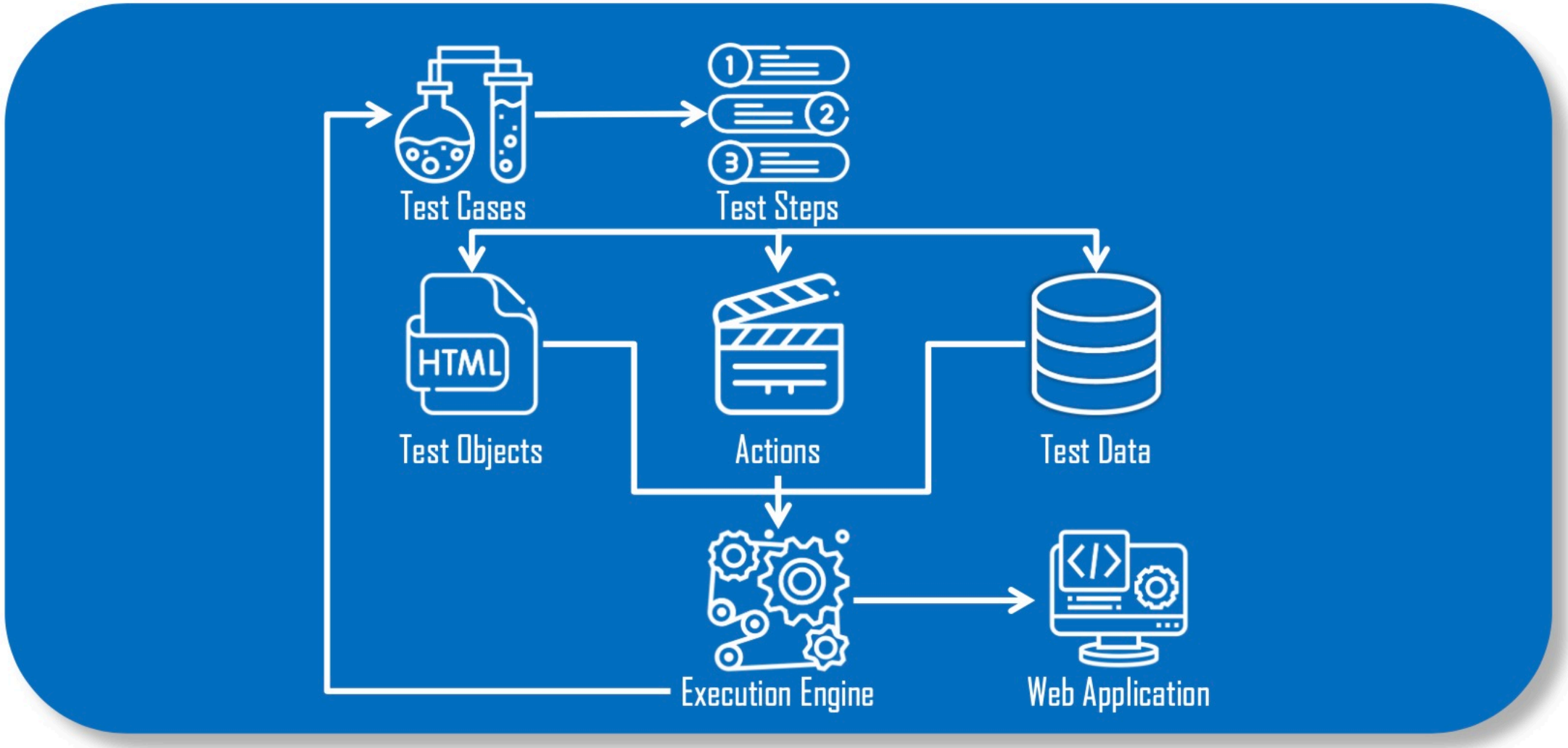
Execution Engine

Also known as the Driver Script, it compiles all the data together and executes it on the application

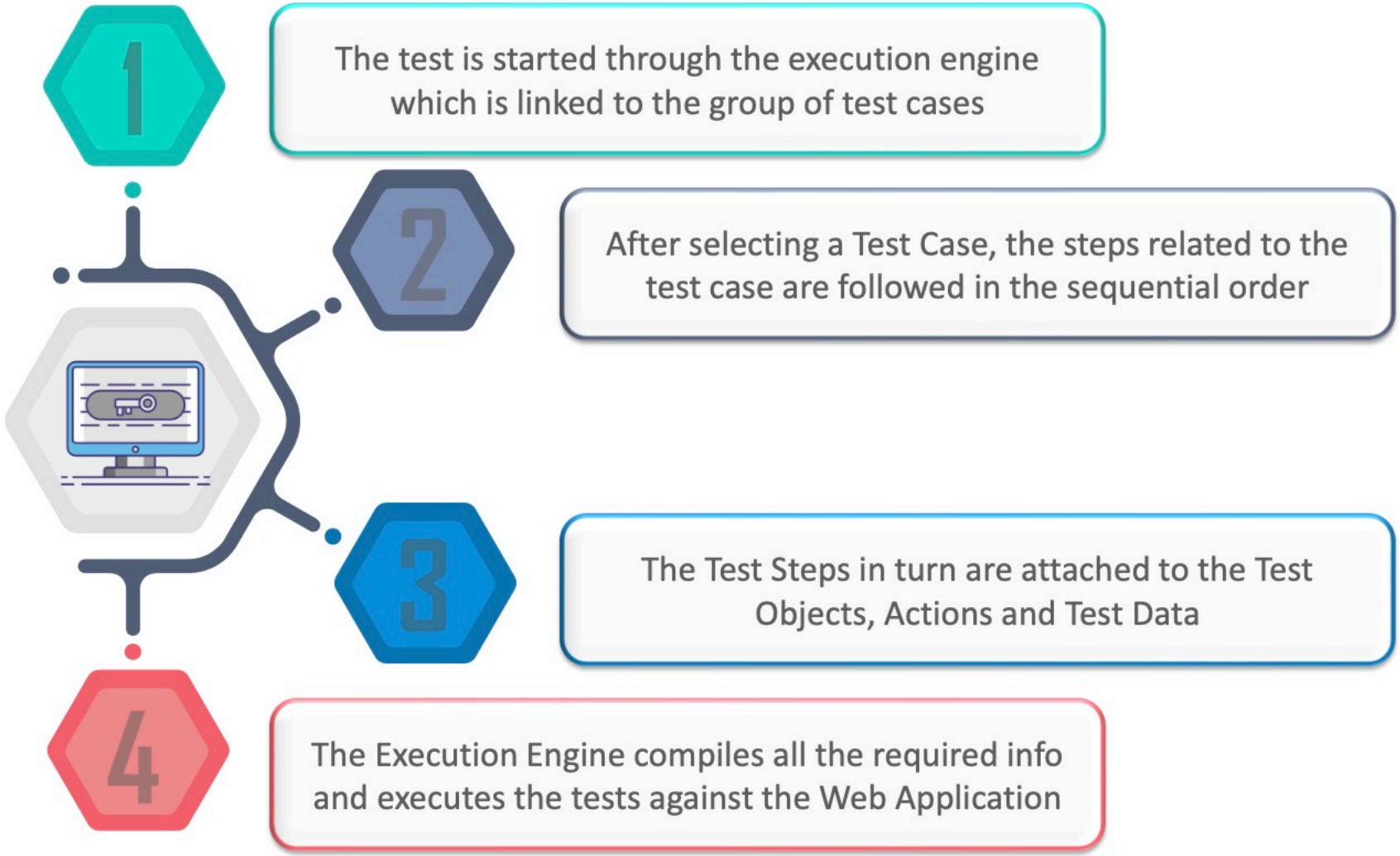
Keyword Function Library

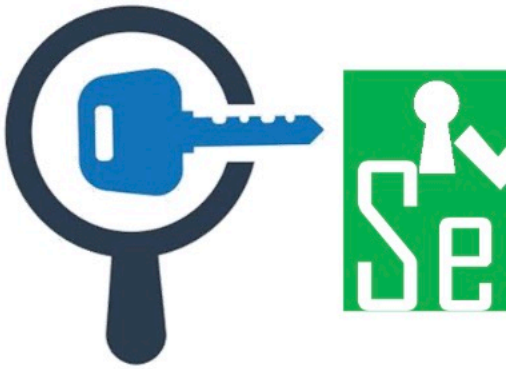
Keyword-function binding library which is tied to the actions performed by the test steps

Keyword-Driven Testing Framework Flow



Keyword-Driven Testing Framework Flow



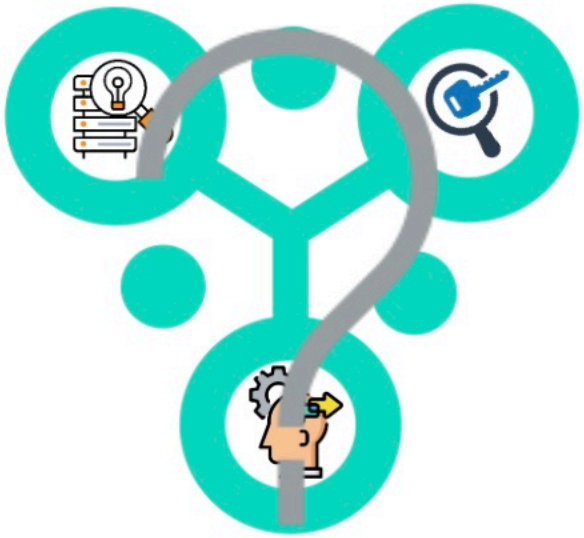


What is Hybrid Framework?

Hybrid Frameworks are set up by combining two or more frameworks together. This is usually done to enhance the existing framework or to mitigate its weaknesses

Why use Hybrid Framework?

- Organizational requirements may change as it grows
- One framework cannot fulfil the testing needs of a changing application
- Hybrid framework suits best for testing dynamic applications



Advantages of Hybrid Framework

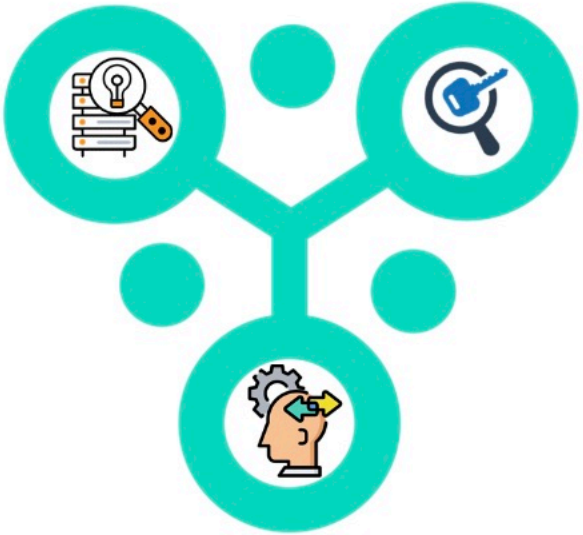
Advantages

Flexibility to work with different frameworks together

Fast and reliable approach for dynamic systems

Higher code reusability, reduced redundancy, and duplication of effort

Better error and window handling



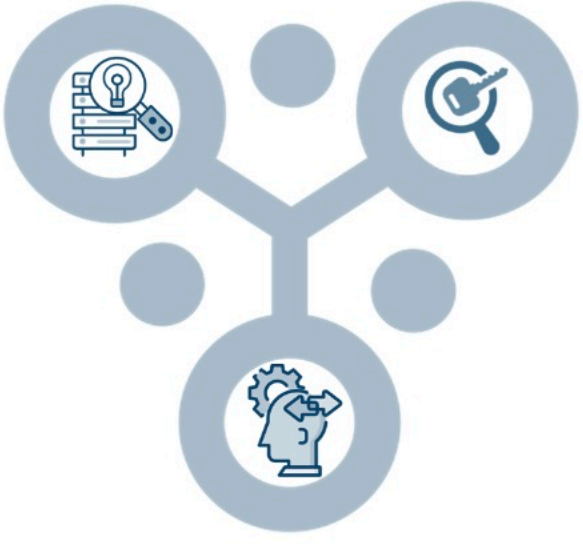
Limitations of Hybrid Framework

Limitations

Complexity increases as new stuff is added to the framework

If not implemented properly, errors stack up more than single framework implementation

Reporting mechanism does not fit easily into the framework



Hybrid Framework Best Practices

Framework should be flexible enough to easily switch formats e.g excel-> JSON

The core framework code should be separate from the test scripts

Should be flexible for new feature additions such as new browser coverage or new locators

Should efficiently utilise the Page Object Model design pattern

Framework should allow addition of new environments such as addition of a new Grid

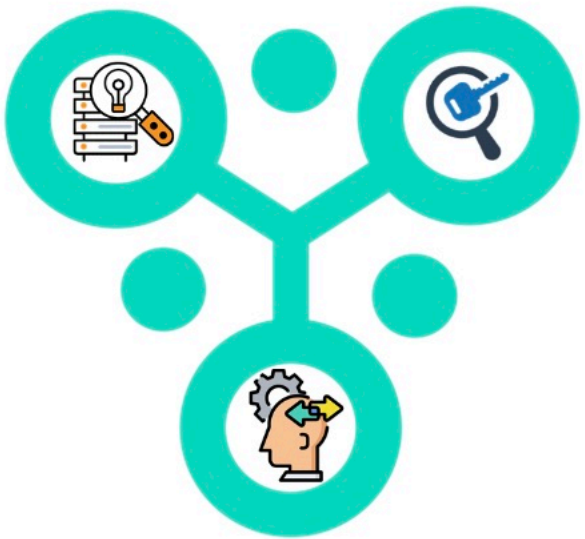
A robust reporting mechanism should be in place

Generic Methods for repetitive tasks should be maintained

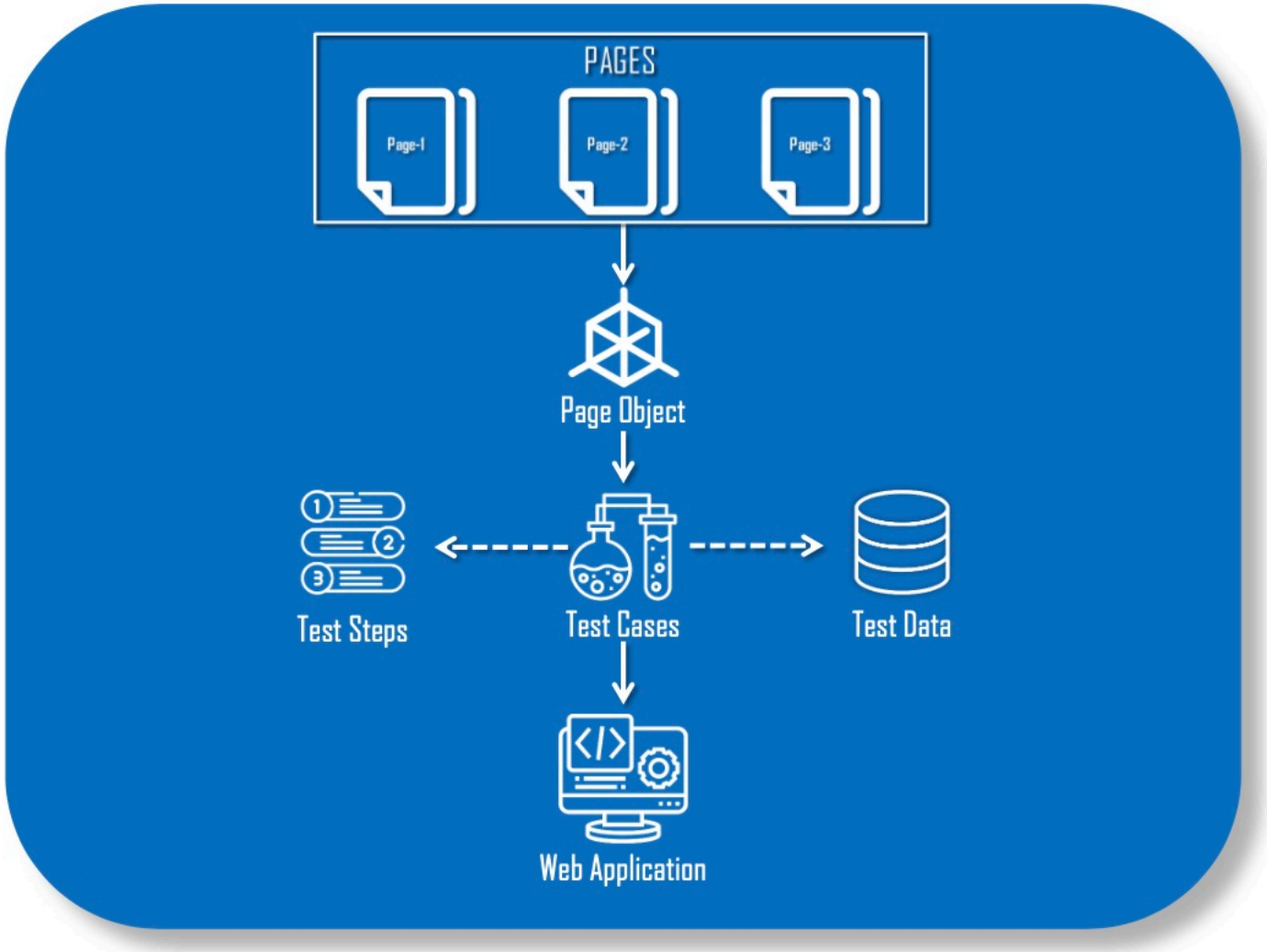
A proper directory structure needs to be followed for the entire framework

Hybrid Framework with Page Object Model

- Page Object Model is utilised to get the most out of the automation framework being used
- The POM design pattern helps to streamline the framework implementation
- Easier for new users to adopt and use the framework directly

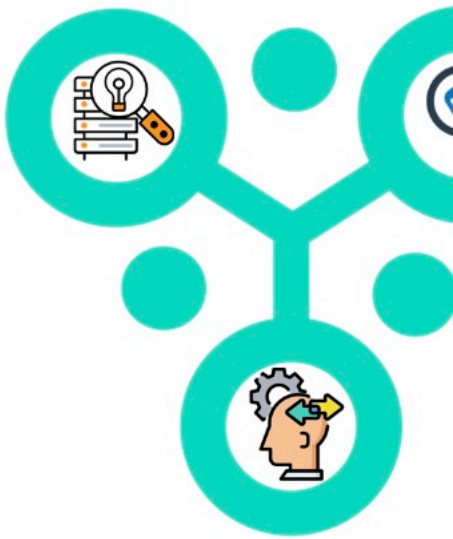


Hybrid Framework with Page Object Model Flow



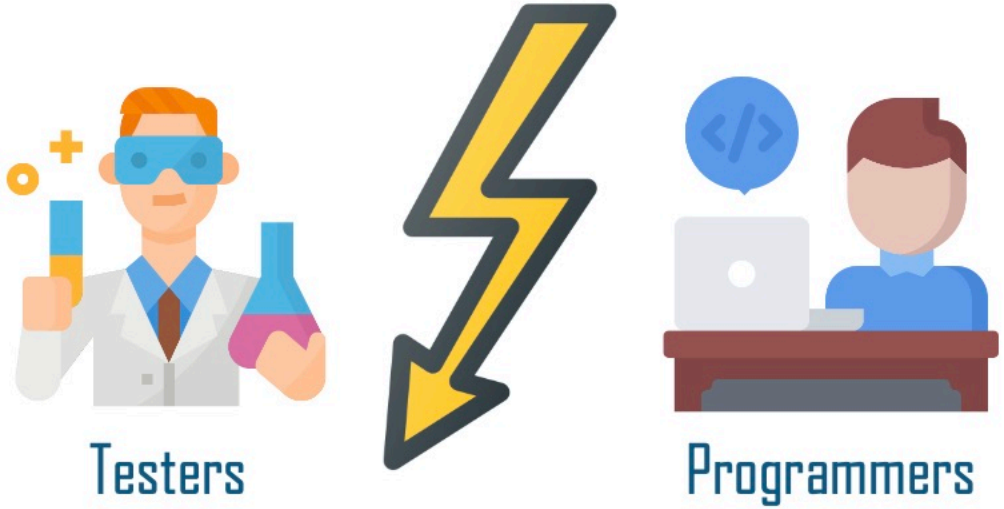
Hybrid Framework with Page Object Model Flow

1. Page library for each unique web-page is created
2. The library is connected to the object repository which further feeds into the test cases
3. To further simplify the process, action items (methods) for each page can be created separately
4. The test scripts are compiled and executed using the available test data and action items



Agile Methodology and Testing

- Traditional testing approach created silos between testers and programmers
- Organizations using the agile development method need their testing environment to be equally agile
- Traditional testing methods sacrificed speed for better quality and vice-versa
- Agile methodology requires full application life-cycle testing with less overhead and documentation



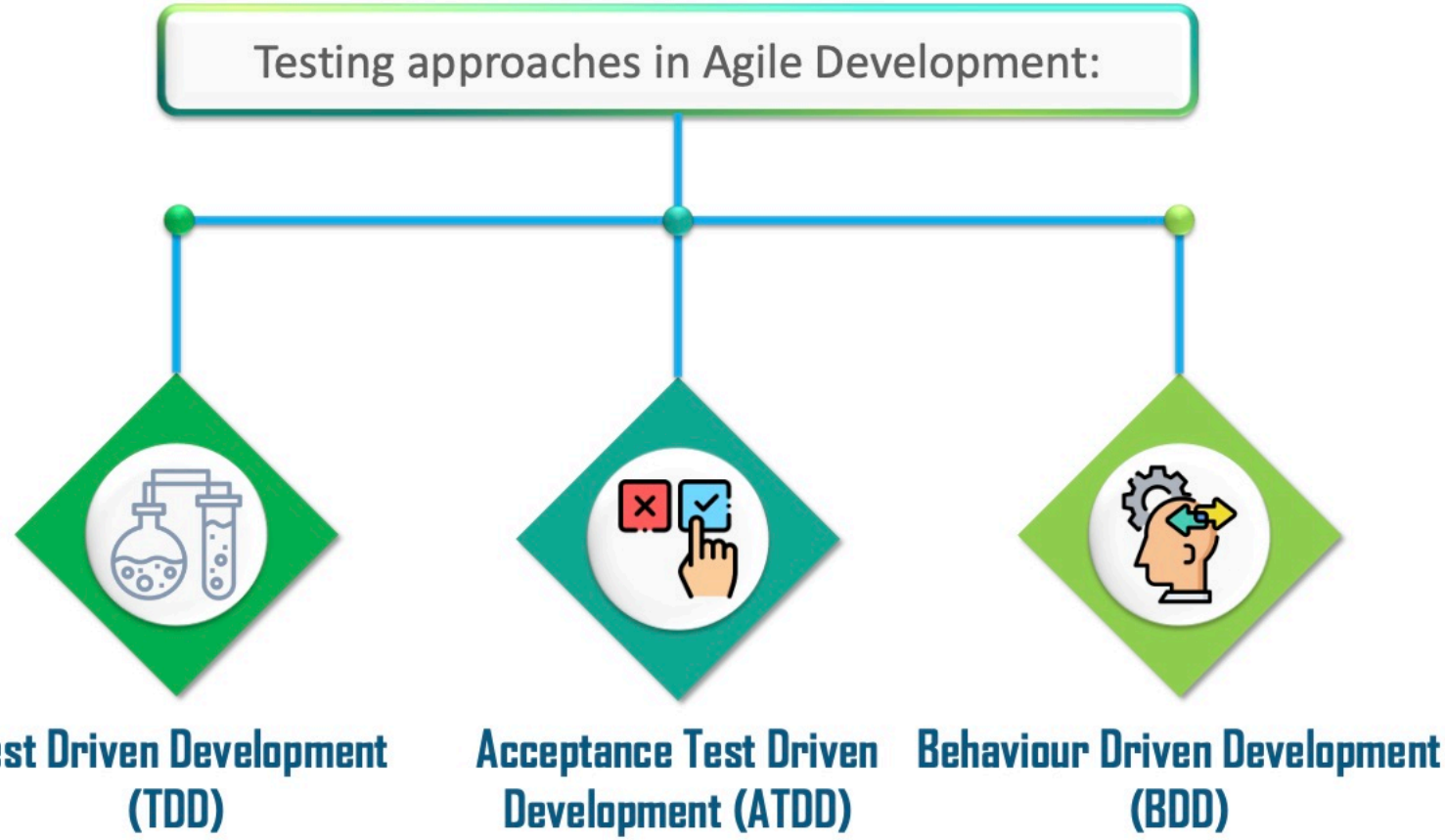
Testing the Agile Way

- Agile testing removes the inefficiencies and the overheads of the traditional testing approach
- It does this by introducing concepts such as test automation, full cycle testing, collaborative testing, etc.
- Utilises approaches like TDD, ATDD, and BDD for better, faster and efficient testing

Testers

Programmers

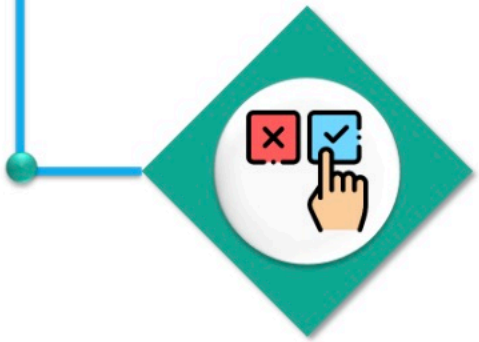
Approaches in Agile Testing



Approaches in Agile Testing

Test Driven Development

- The Development of the project centres around Tests
- Passing all the tests is an important part of the development cycle



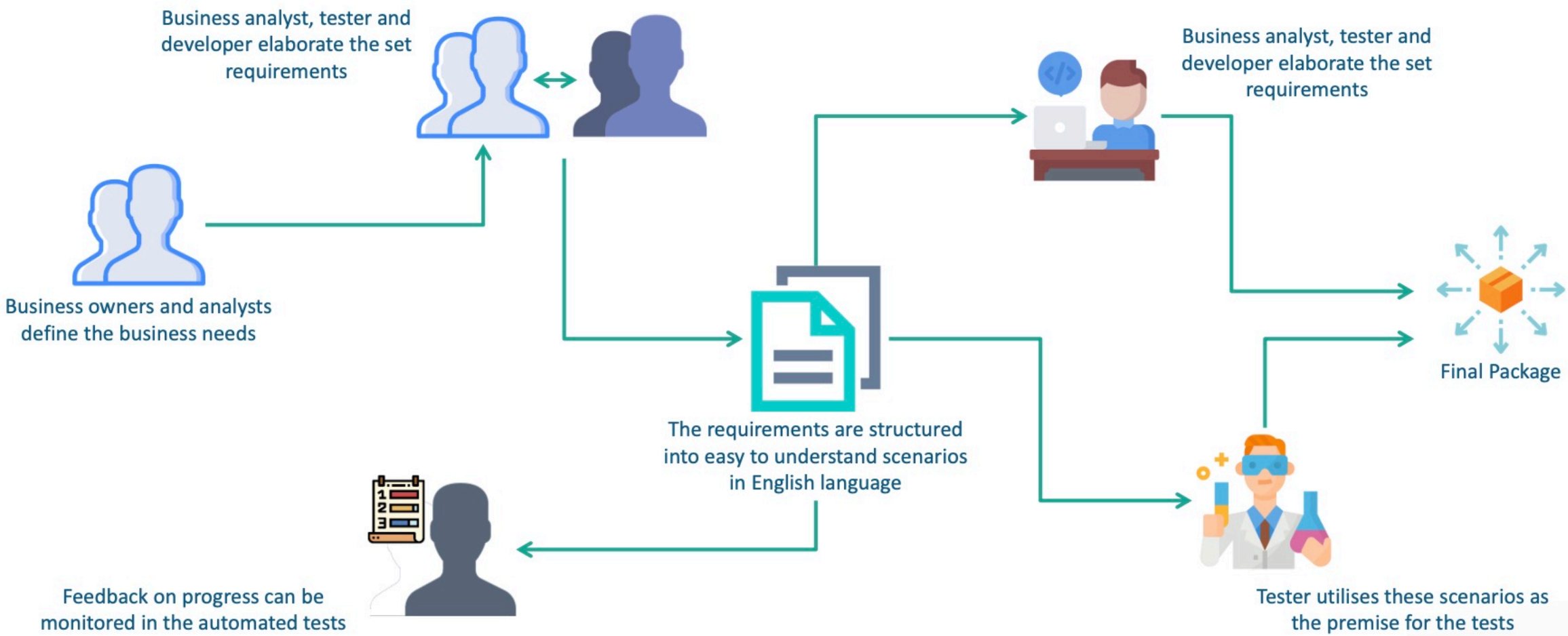
Acceptance Test Driven Development

- Collaborative effort from each team involved in development in writing acceptance tests
- Acceptance tests written in advanced should pass for corresponding functionality

What is Behaviour Driven Development Framework?

- It combines the fundamentals of TDD and ATDD
- Tests are written in a language understood by everyone (Non-Technical personnel)
- Invites inputs from all the teams working on the project for testing requirements

BDD Framework Cycle



Advantages of BDD Framework

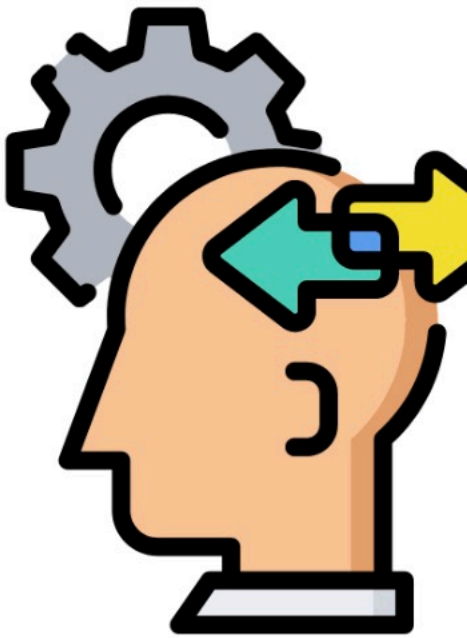
Advantages

Test functions are described in very straight forward and concise manner

Agile principals are followed very closely

Motivates discovery and development of new features

Collaboration between several teams as well as the clients



Limitations of BDD Framework

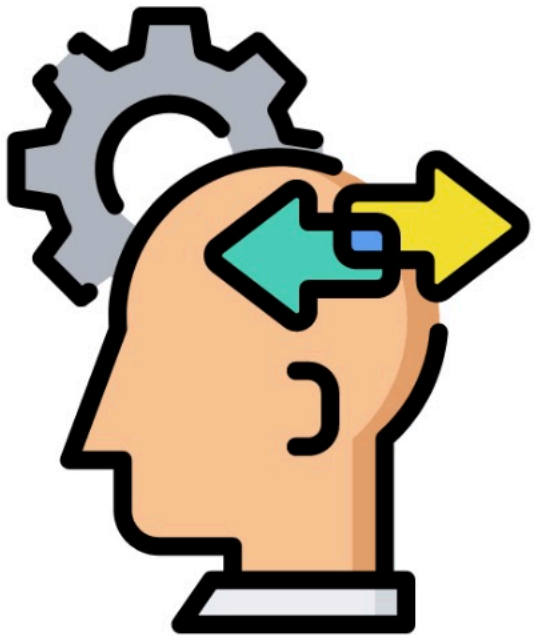
Limitations

If the user/client communication is not there, the process is hampered

Not good for smaller or short term projects

High level of collaboration required from development and testing team

High-availability of client input is required to reduce ambiguity around the project



BDD Testing Framework

BDD Testing Framework utilises Cucumber and Gherkin tools to implement acceptance tests on the web application

Cucumber

- Cucumber is a collaborative tool specifically designed to implement the BDD framework
- It executes plain language specifications and creates reports for the executed tests

Gherkin

- Gherkin syntax is way to define Cucumber specifications
- It is a domain specific language which describes business logic without detailed implementation

Why use Cucumber with Selenium?

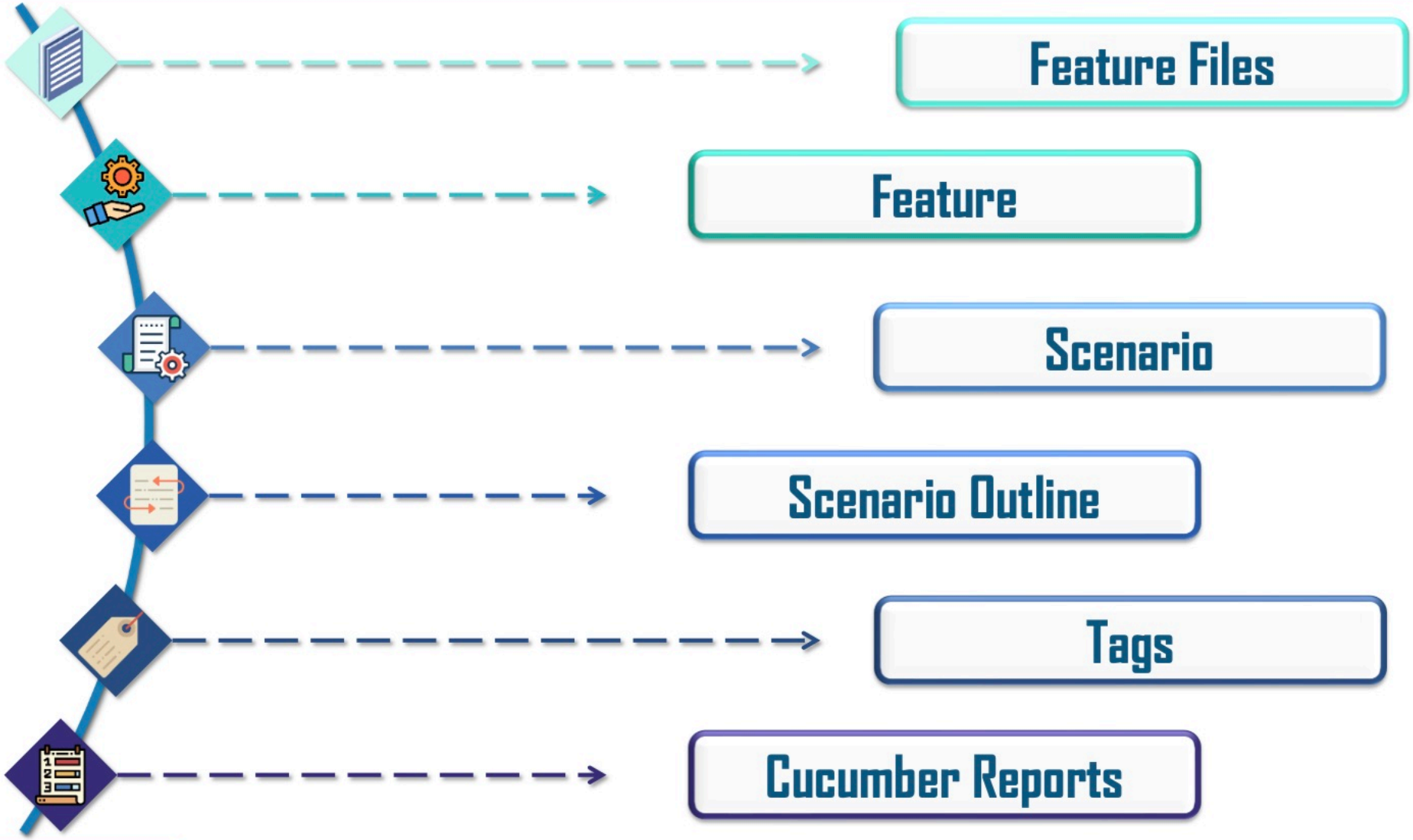
Enables writing tests which can be easily interpreted by various stakeholders

Best way to implement BDD in a Selenium Testing environment

Motivates discovery and development of new features

Reduces quality gap between the developing team and the client

Cucumber Fundamentals



Cucumber Fundamentals

Feature Files

Feature

Scenario

Scenario Outline

Tags

Cucumber Reports

Test Runner

- The most essential part of the Cucumber BDD
- Used for writing automation steps and acceptance tests
- Multiple feature files can be created depending upon use-case
- Feature files are saved with the extension .feature

Sample.feature

```
Feature: feature description
  More Details on the Feature

Scenario: Sample Functionality
  Given Under the travel section
  When User clicks button1
  Then Travel Itinerary is displayed
```

Cucumber Fundamentals

- Feature Files
- **Feature**
- Scenario
- Scenario Outline
- Tags
- Cucumber Reports
- Test Runner

- High-level description of application's particular function
- Should be short, concise and easily understood by everyone
- A single feature file can only contain one feature
- Gherkin parser throws exception if multiple features are in a single feature file

Sample.feature

```
Feature: feature description
  More Details on the Feature

Scenario: Sample Functionality
  Given Under the travel section
  When User clicks button1
  Then Travel Itinerary is displayed
```

Cucumber Fundamentals

- Feature Files
- Feature
- **Scenario**
- Scenario Outline
- Tags
- Cucumber Reports
- Test Runner

- Scenarios depict steps and outcome for a particular test case
- It should be able to explain the intent of the current test case
- Scenarios are written in the Gherkin Syntax
- A feature file can contain multiple scenarios

```
Scenario: Sample Functionality
  Given Under the travel section
  When User clicks button1
  Then Travel Itinerary is displayed

Background:
  Given logged as admin
  And clear last inserted value
```

Cucumber Fundamentals

- Feature Files
- Feature
- **Scenario**
- Scenario Outline
- Tags
- Cucumber Reports
- Test Runner

Given:	Defines the state the application needs to be in before execution. Can be parameterized
When:	Specifies the condition under which the action has to be performed
Then:	Expected outcome of the test is described here
Background:	A repeating step which runs in every scenario. Usually for actions which need to run in the background
And:	Used to add condition or additional details to the step
But:	Adds a negative condition to the testing scenario

Cucumber Fundamentals

- Feature Files
- Feature
- Scenario
- **Scenario Outline**
- Tags
- Cucumber Reports
- Test Runner

- Used when the same test is to be executed multiple times using different data sets
- It fetches values from a tabular structure separated by 'pipe' (||) symbol
- The parameters can be passed in any of the statements under a scenario

```
Scenario Outline: Sample Functionality
  Given search bar is selected on the home page
  When search item is entered as <item>
  Then search results matching the <result> should be displayed

  Examples:
    | item | result|
    | pets | dog  |
    | vehicle| bike|
```

Cucumber Fundamentals

- Feature Files
- Feature
- Scenario
- Scenario Outline
- **Tags**
- Cucumber Reports
- Test Runner

- Helps organizing test scenarios in the test case
- Lets the tester manage and decide which scenario to run during test execution
- Custom tags can be created for each feature file

```
@SampleTag
Feature: feature description
  More Details on the Feature

Scenario: Sample Functionality
  Given Under the travel section
  When User clicks button1
  Then Travel Itinerary is displayed
```

Cucumber Fundamentals

Feature Files

Feature

Scenario

Scenario Outline

Tags

Cucumber Reports

Test Runner

- Cucumber test scenarios result can be accounted using reporting plugins
- Various plugins are built-in and can be used directly:
  - pretty
  - Progress
  - html
- External reporting plugins can also be used

Cucumber Fundamentals

Feature Files

Feature

Scenario

Scenario Outline

Tags

Cucumber Reports

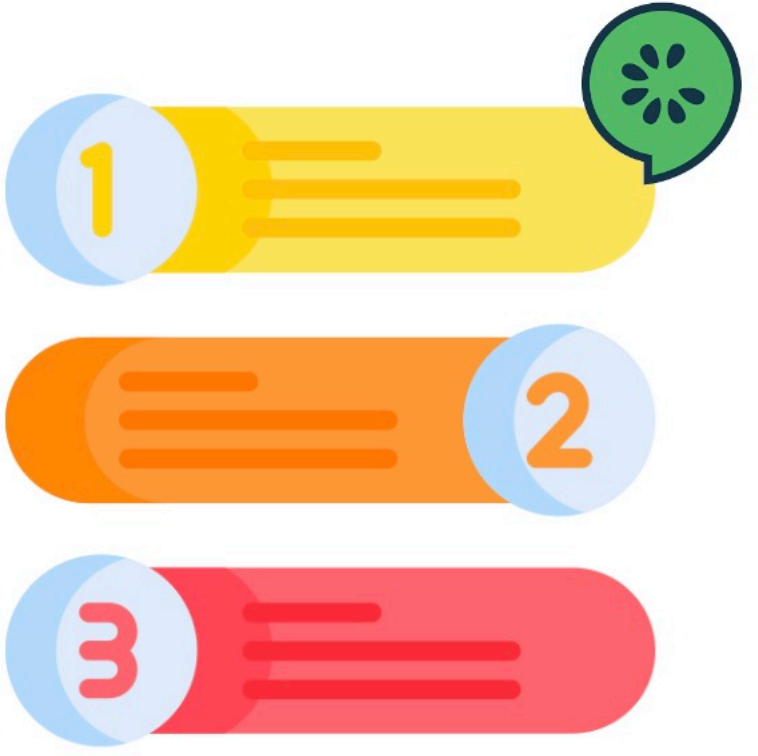
Test Runner

- Supplies the metadata information required for test execution
- Information for feature files, step definition, etc. are passed to the runner class
- Utilises @CucumberOptions and @RunWith() annotations to pass the options
- @RunWith() annotation is a part of the JUnit Framework

```
@RunWith(Cucumber.class)
@CucumberOptions(
        features = "/feature/file/destination"
        ,glue={"src/main/stepDefinition"}
        )

public class TestRunner {

}
```

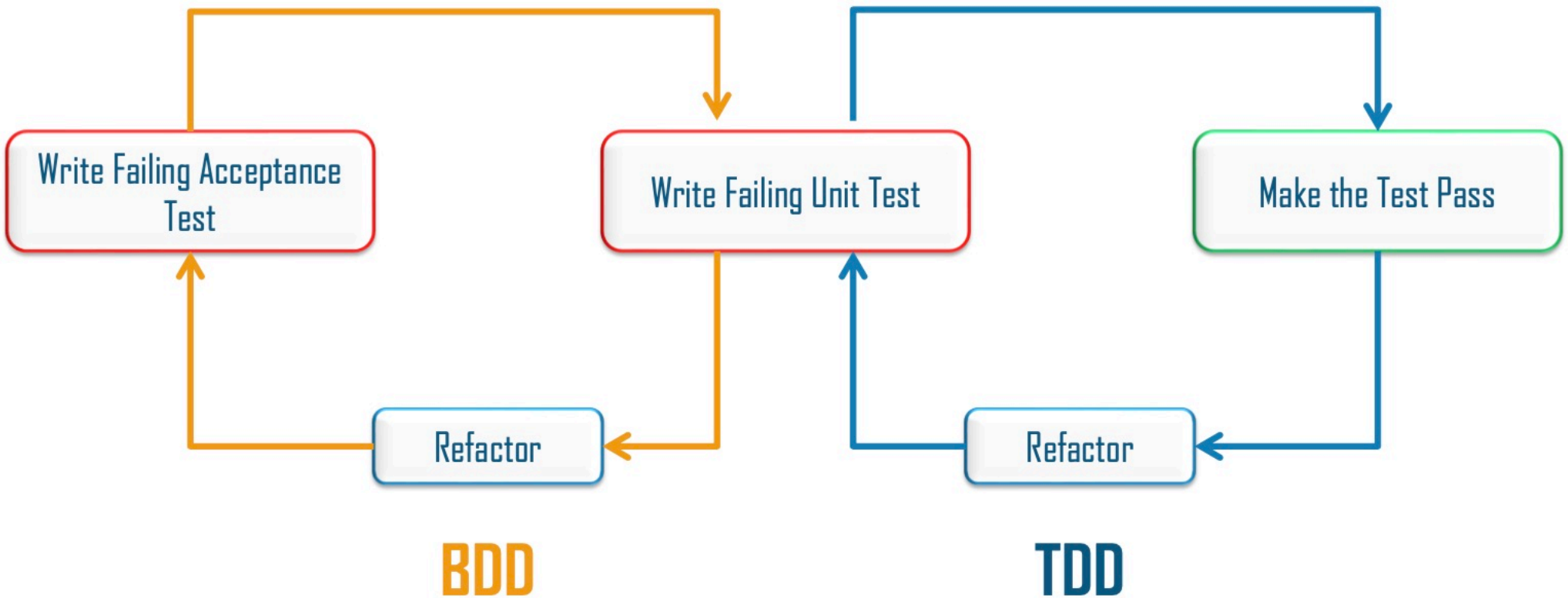
Step Definition

- Step Definition maps the gherkin scenarios in cucumber feature files to the test code
- The actual code is written in the language being used for the project (in this case - java)



Step Definition

```
Scenario: Sample Scenario
 Given checkout page is open
 When payment method is selected
 Then choose credit card option
```

```
public class StepDefinitions{
 @Given("checkout page is open")
  public void givenMethod(){
    …
  }
 @When("payment method is selected")
  public void whenMethod(){
    …
  }
 @Then("choose credit card option")
  public void thenMethod(){
    …
  }
}
```

BDD Framework Testing Cycle