Behavior Driven Development

BDD (Behavioral Driven Development) is a **software development** approach that was developed from **Test Driven Development (TDD)**.

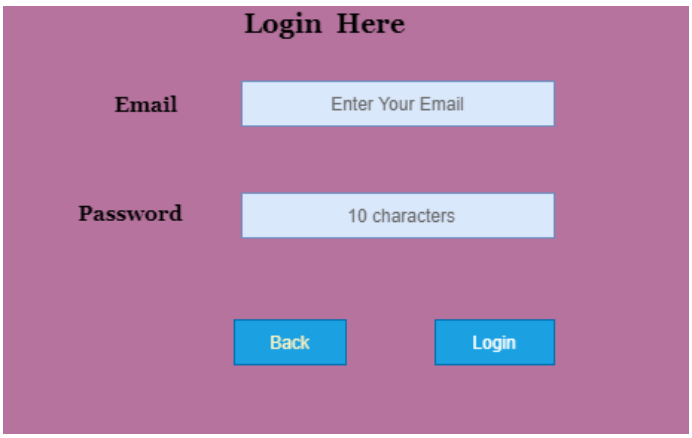
BDD includes test case development on the basis of the behavior of software functionalities. All test cases are written in the form of simple English statements inside a feature file, which is human-generated. Acceptance test case statements are entirely focused on user actions.

BDD is written in simple English language statements, not in a typical programming language. BDD improves communication between technical and non-technical teams and stakeholders.

In the following example, we are going to take the **login function** of a web application.

Example:

In order to ensure the working of Login Functionality, we are developing acceptance test cases on the basis of BDD.



Feature: `Login Function`
`To enter in the System`
`User must be able to`
`Access software when login is successful`
Scenario: `Login`
Given `User has its Email`
And `Password`
When `User enters the correct Email and Password`
Then `It should be logged in`
Scenario: `Unsuccessful Login`
When `User enters either wrong Email or Password`
Then `It should be reverse back on the login page with an error message`

Consider a scenario where you want to test Google Homepage. One of the test scenarios will be to verify that the page displays all the main elements. As part of a test case, let us say that you want to check that the homepage displays the search text box, “Google Search” button and “I’m Feeling Lucky” button. With BDD, you can write this test scenario in the below format:

- **Given** I launch Chrome browser
- **When** I open Google Homepage
- **Then** I verify that the page displays search text box
- **And** the page displays Google Search button
- **And** the page displays Im Feeling Lucky button

Why Cucumber?

There are multiple behaviour-driven development tools such as Cucumber, Concordion, SpecFlow, JDave etc, that let you write your test cases in the format given in the above.

Cucumber is one of the most popular tools because of the reasons given below:

1. Cucumber BDD is open source and hence, its free to use

2. With Cucumber, you can write your test scripts in multiple languages such as Java, Ruby, .NET, Python etc

3. Cucumber easily integrates with Selenium, Ruby on Rails, Watir and other web based testing tools

4. Cucumber is one of the most widely used BDD tools. Due to this, you will find lots of online tutorials and forms to help you with your doubts and queries.

Let us once again have a look at the Google Homepage scenario that we have discussed in the above. Cucumber BDD format as shown in the below image:

GoogleHomepage.feature

```
Feature: Google Homepage
This feature verifies the functionality of Google Homepage

Scenario: Check that main elements on Google Homepage are displayed
Given I launch Chrome browser
When I open Google Homepage
Then I verify that the page displays search text box
And the page displays Google Search button
And the page displays Im Feeling Lucky button
```

The Java equivalent of this scenario is also given below:

GoogleHomepage_PageObject.java

www.automationtestinghub.com

```java
package examples;

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

public class GoogleHomepage_PageObject {

    public static void main(String[] args) {
        //Instantiate web driver
        WebDriver driver = new ChromeDriver();
        driver.get("http://www.google.com");

        //Verify that search text box is displayed on the page
        if(!driver.findElement(By.name("q")).isDisplayed()) {
            System.out.println("Search text box not displayed");
        }

        //Verify that Google Search button is displayed
        if(!driver.findElement(By.name("btnK")).isDisplayed()) {
            System.out.println("Google Search button not displayed");
        }

        //Verify that I'm Feeling Lucky button is displayed
        if(!driver.findElement(By.name("btnI")).isDisplayed()) {
            System.out.println("I'm Feeling Lucky button not displayed");
        }
    }
}
```

You would be thinking as, how the English representation of a test case is related to the actual code.

Basic Terms of Cucumber

- Feature File
- Features
- Tags

- Scenario
- Gherkin Language
- Step Definition

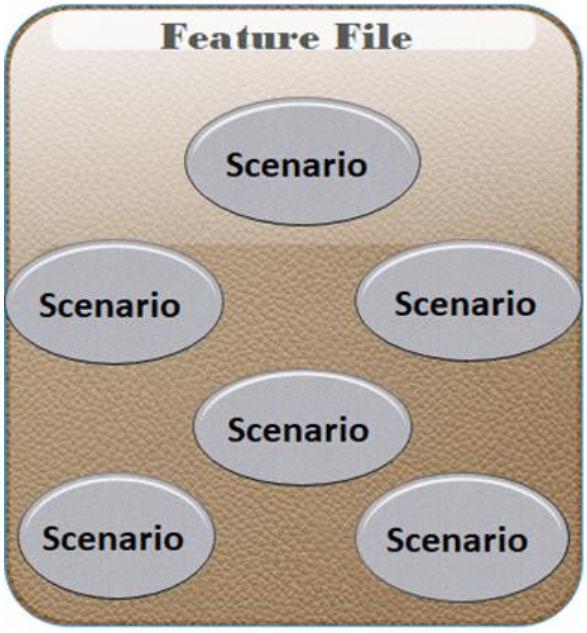
How Cucumber BDD works?

In a Cucumber based setup, there are 3 basic components that you will use in your automation framework. These 3 components are:

- Feature file
- Step definitions file
- Actual code that interacts with the elements on a webpage

1. Feature File

A standalone unit or a single functionality (such as a login) for a project can be called a Feature. Each of these features will have scenarios that must be tested using Selenium integrated with Cucumber. A file that stores data about features, their descriptions, and the scenarios to be tested, is called a **Feature File.**
Cucumber tests are written in these Feature Files that are stored with the extension – **".feature"**. A Feature File can be given a description to make the documentation more legible.



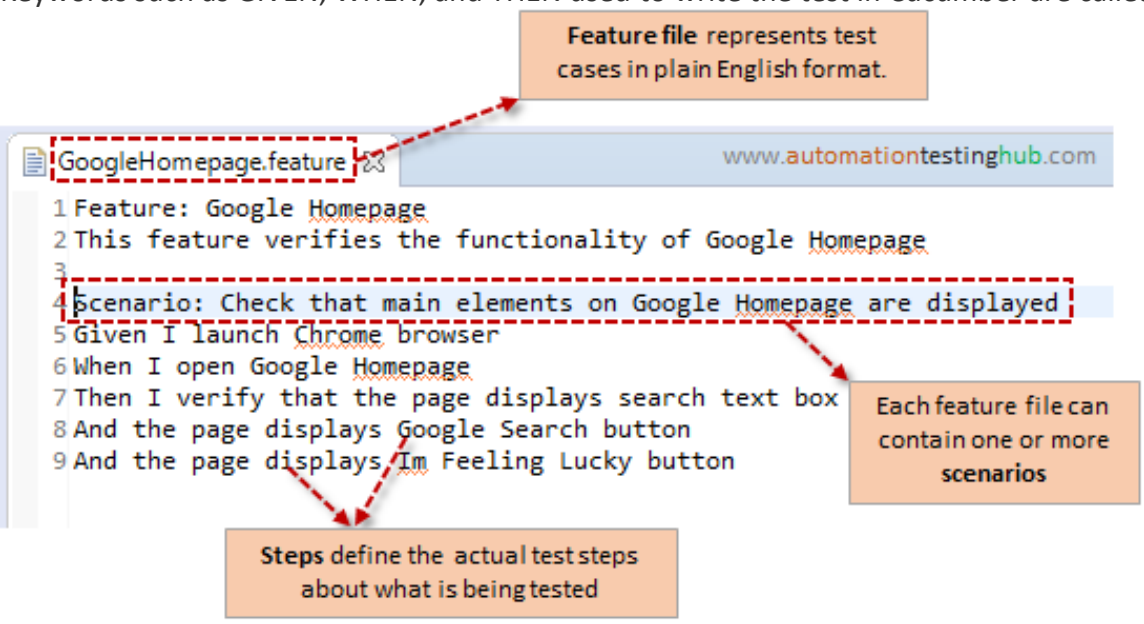
Example:

The Login function on a website

Feature File Name: *googleSearch.feature*

Description: The user shall be able to login upon entering the correct username and password in the correct fields. The user should be directed to the homepage if the username and password entered are correct.

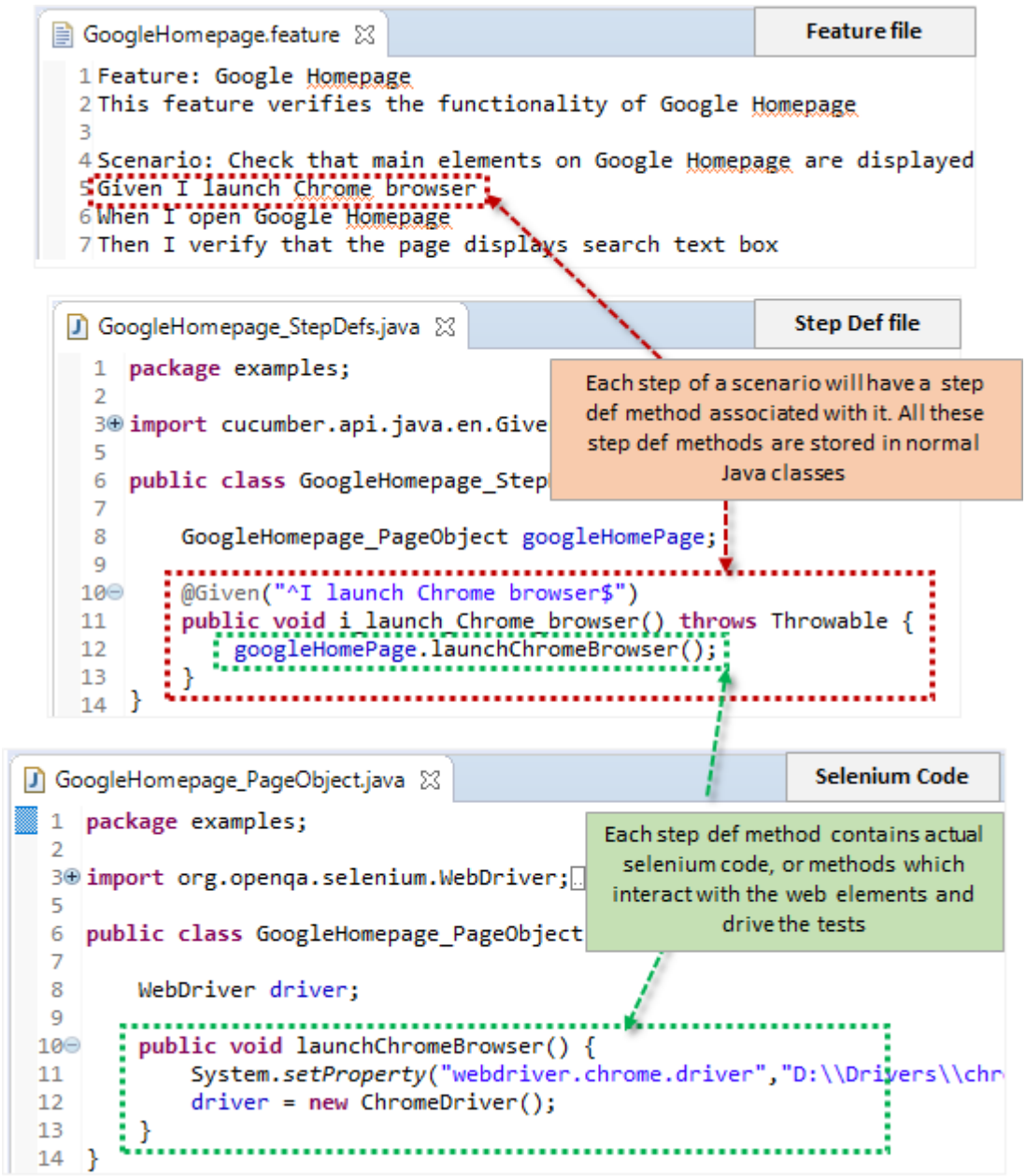
Keywords such as GIVEN, WHEN, and THEN used to write the test in Cucumber are called **Annotations**.



2. Step Definitions file:

Step definitions, or step defs in short, are the methods behind each step in a scenario. For every line or step that you add in a scenario in features file, Cucumber will require a method or a definition using which it knows as to what operations it needs to perform

for that step in the scenario. There is a specific format in which you need to write step defs.



3.Actual code that interacts with web elements:

This is the actual Selenium code that interacts with the webelements and drives your test scripts. You have to write these methods in different classes in the normal way that you do with Selenium. The only additional point here is that you will invoke these methods in the step defs, so that Cucumber knows as to what actual actions it needs to perform as part of a step definition method.

Check this below image which will provide you more clarity on how feature file, step defs and actual selenium code are related.

4. Test Runner File

To run the test, one needs a **Test Runner File**, which is a JUnit Test Runner Class containing the Step Definition location and the other primary metadata required to run the test.

The Test Runner File uses the **@RunWith()** Annotation from JUnit for executing tests. It also uses the **@CucumberOptions** Annotation to define the location of feature files, step definitions, reporting integrations, etc.

Example:

Test Runner Class in cucumberTest package, with the feature files in **"src/test/Feature**" location and Step Definition files in **"src/main/stepDefinition"** folder.

```
package runner;
import cucumber.api.CucumberOptions;
import cucumber.api.testng.AbstractTestNGCucumberTests;
@CucumberOptions(
                    features = {"src/test/resources/features/"},//feature file folder name
                                glue = {"stepdefinitions"},//step definition package name
                               plugin = {"html:target/cucumber-html-report"},// reporting
                                monochrome = true

                    )
public class TestRunner extends AbstractTestNGCucumberTests{

}
```

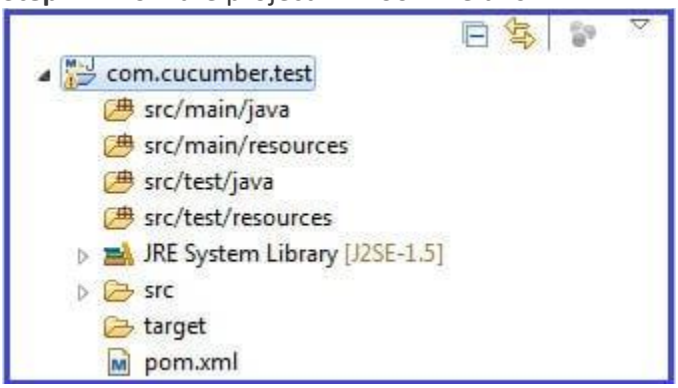
Best Practices in Cucumber Testing

Here are some of the best practices in Cucumber Testing:

- The versions of **Cucumber-java, Cucumber-junit,** and **Cucumber-core** jars should be the same for seamless connectivity.
- Adding an after hook to the code for capturing screenshots when the test fails can help diagnose the issue and debug it.
- Use Tags for organizing tests based on tag definition. This helps in cases where all tests don't have to be run every time. Such tests can be marked using tags and run only when required. This saves time and processing capacity of the system and resources.
- As always, it is important to run the Cucumber Selenium tests on real browsers and devices.

Cucumber Project Setup

Step #1: Create a New Maven Project:
Right Click -> New -> Others -> Maven -> Maven Project -> Next
Step #2: Now the project will look like this:



Step #3: Add below dependencies in pom.xml

```
<dependencies>

<dependency>

    <groupId>info.cukes</groupId>

    <artifactId>cucumber-java</artifactId>

    <version>1.0.2</version>

    <scope>test</scope>

 </dependency>

<dependency>

   <groupId>info.cukes</groupId>

   <artifactId>cucumber-junit</artifactId>

   <version>1.0.2</version>

   <scope>test</scope>

</dependency>

<dependency>

   <groupId>junit</groupId>

   <artifactId>junit</artifactId>

   <version>4.10</version>

   <scope>test</scope>

</dependency>

</dependencies>
```

#1) Feature Files:
Feature files are the essential part of cucumber which is used to write test automation steps or acceptance tests. This can be used as the live document. The steps are the application specification. All the feature files end with .feature extension.

Sample feature file:

Feature: Login Functionality Feature
In order to ensure Login Functionality works,
I want to run the cucumber test to verify it is working

Scenario: Login Functionality
Given user navigates to SOFTWARETETINGHELP.COM
When user logs in using Username as "USER" and Password "PASSWORD"
Then login should be successful
Scenario: Login Functionality
Given user navigates to SOFTWARETETINGHELP.COM
When user logs in using Username as "USER1" and Password "PASSWORD1"
Then error message should be thrown

#2) Feature:

1. **Given:** As mentioned above, given specifies the pre-conditions. It is basically a known state.
2. **When**: This is used when some action is to be performed. As in above example, we have seen when the user tries to log in using username and password, it becomes an action**.**
3. **Then:** The expected outcome or result should be placed here. For Instance: verify the login is successful, successful page navigation.
4. **Background:** Whenever any step is required to perform in each scenario then those steps need to be placed in Background. For Instance: If a user needs to clear database before each scenario then those steps can be put in a background.
5. **And**: And is used to combine two or more same type of action.

Example:

Feature: Login Functionality Feature
Scenario: Login Functionality
Given user navigates to SOFTWARETETINGHELP.COM
When user logs in using Username as "USER"
And password as "password"
Then login should be successful
And Home page should be displayed
Example of Background:
Background:
Given user logged in as databases administrator
And all the junk values are cleared

#3) Scenario Outline:
Scenario outlines are used when the same test has to be performed with different data set. Let's take the same example. We have to test login functionality with multiple different sets of username and password.

Feature: Login Functionality Feature
In order to ensure Login Functionality works,
I want to run the cucumber test to verify it is working

Scenario Outline: Login Functionality
Given user navigates to SOFTWARETESTINGHELP.COM
When user logs in using Username as <**username**> and Password <**password**>
Then login should be successful
Examples:

```
|username      |password      |
|Tom           |password1     |
|Harry         |password2     |
|Jerry         |password3     |
```

Note:

1. As shown in above example column names are passed as a parameter to **When** statement.
2. In place of Scenario, you have to use Scenario Outline.
3. Examples are used to pass different arguments in the tabular format. Vertical pipes are used to separate two different columns. An example can contain many different columns.

#4) Tags:

Cucumber by default runs all scenarios in all the feature files. In real time projects, there could be hundreds of feature file which are not required to run at all times.

For instance: Feature files related to smoke test need not run all the time. So if you mention a tag as smokeless in each feature file which is related to smoke test and runs cucumber test with @SmokeTest tag. Cucumber will run only those feature files specific to given tags. Please follow the below example. You can specify multiple tags in one feature file.

Example of use of single tags:

@SmokeTest
Feature: Login Functionality Feature
In order to ensure Login Functionality works,
I want to run the cucumber test to verify it is working

Scenario Outline: Login Functionality
Given user navigates to SOFTWARETESTINGHELP.COM
When user logs in using Username as <**username**> and Password <**password**>
Then login should be successful
Examples:

```
|username      |password      |
|Tom     |password1     |
|Harry   |password2     |
|Jerry   |password3     |
```

Example of use of multiple tags:

As shown in below example same feature file can be used for smoke test scenarios as well as for login test scenario. When you intend to run your script for a smoke test then use @SmokeTest. Similarly when you want your script to run for Login test use @LoginTest tag.

Any number of tags can be mentioned for a feature file as well as for scenario.

@SmokeTest @LoginTest
Feature: Login Functionality Feature
In order to ensure Login Functionality works,
I want to run the cucumber test to verify it is working

Scenario Outline: Login Functionality
Given user navigates to SOFTWARETETINGHELP.COM
When user logs in using Username as <username> and Password <password>
Then login should be successful
Examples:
username	password
Tom	password1
Harry	password2
Jerry	password3

Similarly, you can specify tags to run the specific scenario in a feature file. Please check below example to run specific scenario.

Feature: Login Functionality Feature
In order to ensure Login Functionality works,
I want to run the cucumber test to verify it is working

@positiveScenario
Scenario: Login Functionality
Given user navigates to SOFTWARETETINGHELP.COM
When user logs in using Username as “USER” and Password “PASSWORD”
Then login should be successful

@negaviveScenario
Scenario: Login Functionality
Given user navigates to SOFTWARETETINGHELP.COM
When user logs in using Username as “USER1” and Password “PASSWORD1”
Then error message should throw

Feature File 1:

```
Feature: Registration
  Background:
  Given user on the homepage
  And user follows "Sign in"
  @regression
  Scenario: Create a New User
  When user fills "registration email textbox" with "chitrali.sharma27@gmail.com"
  And user clicks "create an account button"
  And user enters the following details
  | First Name | Chitrali|
  | Last Name | Sharma|
  | Password | Inquiry@1234 |
  | Date | 17| | Month | 02| | Year | 1992 |
  And user clicks "register button"
  Scenario: User does not follow form validations
  When user enters wrong characters
  Then error message displayed with invalid password
  And user returns back on registration page
```

Feature File 2:

```
Feature: Login
Background:
Given user on the login page
And user follows "Log in"

@regression @smoke
Scenario: Verification of Login Function
Given user on the Login Page
And user enters "email address" with "chitrali.sharma27@gmail.com"
And user enters "password" with "Inquiry@1234"
And user click "log in" button
Then user should see "My Account"
Scenario: Unsuccessful login
Given user on the Login Page
And user enters "email address" with "chitrali.sharma27@gmail.com"
And user enters "password" with "qsder@1234"
And user clicks "login" button
Then error message displayed with wrong password
And user returns back on login page
```