**BDD (Behavior Driven Development) Framework**

BDD framework i.e. Behavior Driven Development is a software development approach that allows the tester/business analyst to create test cases in simple text language (English).

The simple language used in the scenarios helps even non-technical team members to understand what is going on in the software project. This helps and improves communication among technical and non-technical teams, managers, and stakeholders.

**What Is BDD Behavior Driven Development?**

BDD transpires from the TDD i.e. Test Driven Development which allows the users to work with multiple test data with minimum intervention in the software code and thereby helps to increase the reusability of the code, which is a time-saving mechanism in software development/ test automation.

**By inheriting TDD, BDD also has all those features along with its advantages.**

* Test scenarios are written separately in a different file, named as Feature file.
* Tests are written by focusing user stories and system behavior in a layman language.
* Code is subject to be written differently in step definitions file i.e. Java, Python.

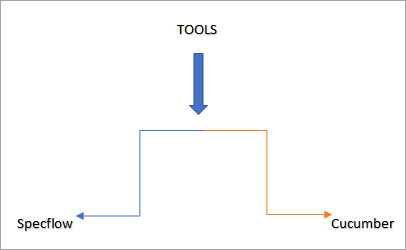
**Why Use BDD Framework?**

Before the BDD framework, everyone was using TDD. TDD works fine in software development, provided the stakeholders are familiar with the framework being used and their technical knowledge is sufficient. However, this may not be the case always.

BDD provides a path that acts as a bridge to overcome the gap between the technical and the non-technical teams because the test cases are commonly written in simple text, i.e. English. The main advantage of BDD is the low jargon and clearer approach which is easier to understand.

**How to Implement the BDD Approach?**

Test scenarios should be written in plain language with a detailed description of the test, how to test the application and the application behavior which can be understandable by all.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/07/BDD-approach.png)

In this tutorial, we will focus on [Cucumber](https://cucumber.io/) – a software tool for BDD and will learn to practically implement it using its language i.e. Gherkin.

**Cucumber – A BDD Framework Tool**

Cucumber is a Behavior Driven Development (BDD) framework tool to write test cases.

**Given – When – Then Approach**

* **Given:** Some given context (Preconditions).
* **When:**Some Action is performed (Actions).
* **Then:**Particular outcome/consequence after the above step (Results).

**Sample Feature File**

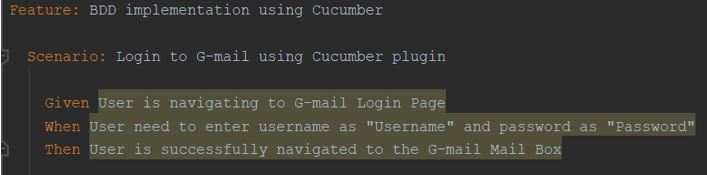
**Feature**: BDD implementation using Cucumber

**Scenario**: Login to G-mail using Cucumber plugin

**Given** User is navigating to G-mail Login Page

**When** User need to enter username as "Username" and password as "Password"

**Then** User is successfully navigated to the G-mail Mail Box

****

**Sample Step Definition File**

|  |
| --- |
| **import** cucumber.api.java.en.Given;  **import** cucumber.api.java.en.Then;  **import** cucumber.api.java.en.When;    **public** **class** Sample {      @Given("^User is navigating to G-mail Login Page$")  **public** **void** user\_is\_navigating\_to\_G\_mail\_Login\_Page() **throws** Throwable {          // Write code here that turns the phrase above into concrete actions        }     @When("^User need to enter username as \"([^\"]\*)\" and password as \"([^\"]\*)\"$")  **public** **void** user\_need\_to\_enter\_username\_as\_and\_password\_as(String arg1, String arg2) **throws** Throwable {          // Write code here that turns the phrase above into concrete actions        }      @Then("^User is successfully navigated to the G-mail Mail Box$")  **public** **void** user\_is\_successfully\_navigated\_to\_the\_G\_mail\_Mail\_Box() **throws** Throwable {          // Write code here that turns the phrase above into concrete actions        }  } |

Cucumber is a test plugin that helps in the implementation of the behavior-driven development approach.

**Advantages Of BDD Framework**

Enlisted below are the various advantages of BDD.

**#1) Coverage of User Stories**

Hybrid Framework with BDD is meant to be combined with different features. Every resource in the software development phase can contribute to the BDD framework.

Due to its easy concept of layman text in the form of feature file allows the stakeholders of technical resources to write the scenarios in Gherkin language using the user stories. The compatibility of the plain text helps to gain maximum coverage on testing.

**Feature file containing scenarios are:**

* Defined user stories from the business.
* Criteria for the developers to determine if specifications meet business requirements.
* Test scenarios for the testing team.
* Shell cover for an automation tester which allows them to separately write their code in step definition files.
* Explained test scenarios for Stakeholders.

The classification of the step definitions helps the automation tester to keep his code untouched which thereby helps in the maintenance of the scripts.

**#2) Clarity of Scenarios**

Gherkin language uses plain layman text that is focused on the outcome of the product which is being tested/developed using BDD.

As feature file separate the technical description in a different step definitions file for automation testers, it smartly helps a non-technical person to understand the automated test easily. Any updates can be implemented in a small discussion.

Readability power of gherkin guarantees the clarity of scenarios to each of its user which in turn, helps in building the right product.

**#3) Automation of Test Scenarios**

Cucumber implementation in a BDD framework allows an automation tester to easily initiate the scripting with the right approach. Easy language of cucumber scenarios helps them to understand the functionality in a better way.

Cucumber is a language-independent plugin as it is compatible with many programming languages **E.g.** [Java](https://www.softwaretestinghelp.com/java/), [Python](https://www.softwaretestinghelp.com/python/), etc.

**#4) Code Reuse in Framework**

Given – When – Then approach gives liberty to the testers to use the same steps as many times we want in the feature file which gradually helps in saving time for the automation testers.

**Example:**

**Scenario: Scenario 1**

**Given***User is navigated to Google Home Page*  
**When***User searched “Cucumber” in the search engine*  
**Then***Clicked on the Search Button*  
**And***User can see search results for Cucumber in the web browser*

**Scenario: Scenario 2**

**Given***User is navigated to Google Home Page*  
**When***User searched “Selenium” in the search engine*  
**Then***Clicked on the Search Button*  
**And***User can see search results for Selenium in the web browser*

In the above two scenarios, we can conclude that “*Given”,* “*When*” and “*Then*” steps are reusable in the second scenario.

**#5) Parameterization in Feature File**

A user can parameterize the gherkin steps in the feature file to obtain reusability in the file.

**For Example,** if a user is working on a bank application where he logs in to the application again and again. Such kind of steps could be parameterized with a different set of data and it saves time for the tester.

While writing the scenarios, the user has to define the feature file steps in such a way, so that the user can use the common functionality easily.

**#6) Continuous Integration – Easy to Integrate**

Cucumber also supports working with Jenkins. You can run the cucumber test execution in Jenkins and also implement the same in Jenkins slave machines. The cucumber reporting plugin also provides users with an expanded view to track test scenarios.