**What is TDD ?**

🡪**TDD** is an iterative development process. Each iteration starts with a set of tests written for a new piece of functionality. These tests are supposed to fail during the start of iteration as there will be no application code corresponding to the tests. In the next phase of the iteration, Application code is written with an intention to pass all the tests written earlier in the iteration. Once the application code is ready tests are run.

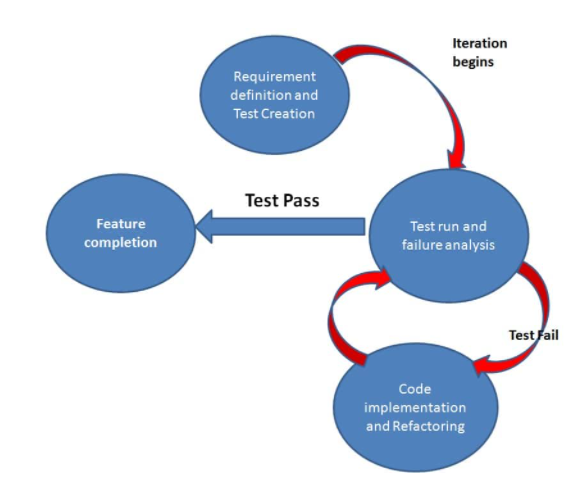
Any failures in the test run are marked and more Application code is written/re-factored to make these tests pass. Once application code is added/re-factored the tests are run again. This cycle keeps on happening until all the tests pass. Once all the tests pass we can be sure that all the features for which tests were written have been developed.

**Benefits of TDD**

1. Unit test proves that the code actually works
2. Can drive the design of the program
3. Refactoring allows improving the design of the code
4. Low-Level regression test suite
5. Test first reduce the cost of the bugs

**Drawbacks of TDD**

1. Developer can consider it as a waste of time
2. The test can be targeted on verification of classes and methods and not on what the code really should do
3. Test become part of the maintenance overhead of a project
4. Rewrite the test when requirements change.



**What is BDD ?**

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

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

This difference brings in the need to have a language that can define, in an understandable format.

## Features of BDD

1. Shifting from thinking in "**tests**" to thinking in "**behavior**"
2. Collaboration between Business stakeholders, Business Analysts, QA Team and developers
3. Ubiquitous language, it is easy to describe
4. Driven by Business Value
5. Extends Test-Driven Development (TDD) by utilizing natural language that non-technical stakeholders can understand
6. BDD frameworks such as Cucumber or JBehave are an enabler, acting a "**bridge**" between Business & Technical Language

BDD is popular and can be utilised for **Unit level** test cases and for **UI level** test cases. Tools like **RSpec** (for Ruby) or in .NET something like **MSpec** or **SpecUnit** is popular for Unit Testing following BDD approach.  Alternatively, you can write BDD-style specifications about **UI interactions**. Assuming you’re building a web application, you’ll probably use a browser automation library like **WatiR/WatiN or Selenium**, and script it either using one of the frameworks I just mentioned, or a given/when/then tool such as **Cucumber (for Ruby)** or **SpecFlow (for .NET)**.

## BDD Tools Cucumber

## What is Cucumber?

**Cucumber** is a testing framework which supports **Behavior Driven Development (BDD)**. It lets us define application behavior in plain meaningful English text using a simple grammar defined by a language called **Gherkin**. Cucumber itself is written in **Ruby**, but it can be used to “**test**” code written in *Ruby* or other languages including but not limited to *Java*, *C#* and *Python*.

## Why BDD Framework?

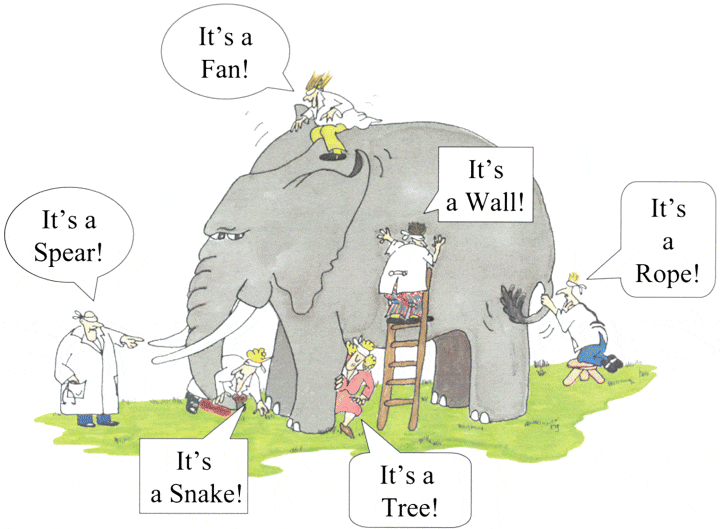
Let's assume there is a requirement from a client for an E-Commerce website to increase the sales of the product with implementing some new features on the website. The only challenge of the development team is to convert the client idea into something that actually delivers the benefits to client.

The original idea is awesome. But the only challenge here is that the person who is developing the idea is not the same person who has this idea. If the person who has the idea happens to be a talented software developer, then we might be in luck: the idea could be turned into working software without ever needing to be explained to anyone else. Now the idea needs to be communicated and has to travel from Business Owners(Client) to the development teams or many other people.

Most software projects involve teams of several people working collaboratively together, so high-quality communication is critical to their success. As you probably know, good communication isn’t just about eloquently describing your ideas to others; you also need to solicit feedback to ensure you’ve been understood correctly. This is why agile software teams have learned to work in small increments, using the software that’s built incrementally as the feedback that says to the stakeholders “***Is this what you mean?***”

Below image is the example of what clients have in their mind and communicated to the team of developers and how developers understands it and work on it.

### ****Wrong Perception****



With the help of Gherkin language cucumber helps facilitate the discovery and use of a ubiquitous language within the team. Tests written in cucumber directly interact with the development code, but the tests are written in a language that is quite easy to understand by the business stakeholders. Cucumber test removes many misunderstandings long before they create any ambiguities in to the code.

### *****Example of a Cucumber/SpecFlow/BDD Test:*****

The main feature of the Cucumber is that it focuses on Acceptance testing. It made it easy for anyone in the team to read and write test and with this feature it brings business users into the test process, helping teams to explore and understand requirements.

***Feature: Sign up***

***Sign up should be quick and friendly.***

***Scenario: Successful sign-up***

***New users should get a confirmation email and be greeted personally by the site once signed in.***

***Given I have chosen to sign up***

***When I sign up with valid details***

***Then I should receive a confirmation email***

***And I should see a personalized greeting message***

***Scenario: Duplicate email***

***When someone tries to create an account for an email address that already exists.***

***Given I have chosen to sign up***

***But I enter an email address that has already registered***

***Then I should be told that the email is already registered***

***And I should be offered the option to recover my password***

Now take a look at the above example code anybody can understand the working of the test and what it is intended to do. It gives an unexpected powerful impact by enabling people to visualize the system before it has been built. Any of the business users would read and understand the test and able to give you feedback that whether it reflects their understanding of what the system should do, and it can even lead to thinking of other scenarios that need to be considered too.

## What is Gherkin - BDD Language?

Before diving into Gherkin, it is necessary to understand the importance and need of a common language across different domains of project. By different domains I  By different domains I mean **Clients, Developers, Testers, Business analysts and the Managerial** team. Let's start by talking about usual problems of a development project first and then we will move to a solution, while doing so we will come across the need for a common language.

Assume you are a part of a technical team (*Developer and Tester*) and you have a task of collaborating with the business team (*Business owners and Business analysts*). You have to come up with the requirements of your project, these requirements will be what your development team will be implementing and test team will be testing. Also, that you have to make a small search feature on your E-Commerce platform. This feature will allow users to search for a product on your website.

As we all might have faced in our experience that requirement given by business team are very crude and basic. For example, in this scenario we may get the following requirements:

**3.    Functional Requirements**

**3.1    Search Functionality**

3.1.1     User should be able to search for a product

3.1.2    Only the products related to search string should be displayed.

### Questions raised from the above requirements

As we can see these requirements are good and useful but are not accurate. They describe a broad behavior of the system but do not specify concrete behavior of the system. Let me illustrate it by dissecting the first requirement, first requirement says that user should be able to search for a product but it fails to specify following

*- What is the maximum searchable length of search string?*

*- What should be the search results if user searches for an invalid product?*

*- What are the valid characters that can be used to search?*

and similarly a few more detailed behavior of the application.

Usually in a project we end up asking above questions with the business team and we get replies, most of the replies reach the project documentation but the unfortunate ones are lost in emails and telephonic conversations. Also these replies are open to interpretation, for example:

**Question to Business Owner***: What should be the search results if user searches for an invalid product?*

**Reply from Business Owner** : *Invalid product searches should show following text on the search page:***No product found**

### Answers of the Questions result in to more Doubts and Interpretation

We get the answers of the questions asked from the Business team but it opens for interpretation or doubts in following ways:

*- Definition of invalid product is ambiguous and different team members will interpret it in different ways. One may consider that an invalid product is one which is not present in the inventory and other team member might consider an invalid product to be one which is a spelling mistake.*

*- The answer by the business team says that "***No product found***" text should be displayed on the page. Does it says that a new search option should be present for the user? or may be related/similar search options should be displayed for the user?*

These are exact points where error is introduced in the system. Also, if we analyze the second doubt we would see that user Business team would love to have a new search option and related/similar searches option presented to the user. However, they were not able to think of this scenario when the question was asked. As a result what happened in the above example is

*1. Business team and the technical teams are communicating at two different levels, business team being vague and technical team trying to be precise.* *2. Ambiguity being introduced in the system, here by the definition of "***invalid product***".*

*3. Not enough insight being given to the Business team, so that they could have come up with new scenarios.*

*4. Some details of project being lost in emails and telephonic conversations.*

### How to Improve the Requirement?

Now let's improve the first requirement given by the business team and try to make it more precise:

"**When a user searches, without spelling mistake, for a product name present in the inventory. All the products with similar name should be displayed**"

"**When a user searches, without spelling mistake, for a product name present in the inventory. Search results should be displayed with exact matches first and then similar matches**"

Here we can see that how clear the requirements have become and with these clear requirements we are able to think more about the system. For eg. In the case of second requirement, after reading it we may think of other scenarios like:

* *What should happen when there no exact and similar matches?*
* *Should the user be given an error message?*
* *Or the user is given a message stating when the product is expected to arrive in inventory.*

### What have we achieved here?

We have forced the client to think in terms of details. With this improved thinking Business teams are coming with more refined requirements. This in turn with reduces the ambiguity in the project and will make developers and testers life easy by reducing the number of incorrect implementations. Also, you can see that each requirement now documents one exact behavior of the application. This means that it can be considered as a requirement document in itself.

### What's the conclusion?

Well, with the above example or exercise we can conclude the followings:

1. *Different teams in the project need a common language to express requirements. This language should be simple enough to be understood by Business team members and should be explicit enough to remove most of the ambiguities for developers and testers.*
2. *This language should open up the thinking of team members to come up with more scenarios. As you express more details you try to visualize the system more and hence you end up making more user scenarios.*
3. *This language should be good enough to be used as project documentation.*

To answer these problems **Gherkin** was created. Gherkin is a simple, lightweight and structured language which uses regular spoken language to describe requirements and scenarios. By regular spoken language we mean English, French and around 30 more languages.

**Example of Gherkin**

As Gherkin is a structured language it follows some syntax let us first see a simple scenario described in gherkin.

**Feature: Search feature for users This feature is very important because it will allow users to filter products**

**Scenario: When a user searches, without spelling mistake, for a product name present in inventory. All the products with similar name should be displayed**

**Given User is on the main page of www.myshopingsite.com**

**When User searches for laptops**

**Then search page should be updated with the lists of laptops**

Gherkin contains a set of keywords which define different premise of the scenario. As we can see above the **colored** parts are the keywords. We will discuss about the gherkin test structure in details later but the key points to note are:

* *- The test is written in plain English which is common to all the domains of your project team.*
* *-This test is structured that makes it capable of being read in an automated way. There by creating automation tests at the same time while describing the scenario.*

**How to use BDD ?**

**- Softwares to Download**

1. Eclipse plugin for Cucumber

Steps to download :Click on Help 🡪 Eclipse Market Place 🡪 In Search Bar “**Eclipse plugin for cucumber**” 🡪Enter🡪 Install

2. Natuarls Plugin

Steps to download :Click on Help 🡪 Eclipse Market Place 🡪 In Search Bar “**Naturals plugin**” 🡪Enter🡪 Install

Alternative Way to download Naturals Plugin

update site: http://rlogiacco.github.io/Natural - http://rlogiacco.github.io/Natural

Watch the below Video

<https://www.youtube.com/watch?v=aOxhKAi3ryA>

**Dependencies**

* **Cucumber Java**
* **Cucumber Core**
* **Cucumber TestNg**
* **Cucumber Java Deps**
* **Selenium Java**

**Cucumber Keywords**

* **Feature file**
* **Scenario**
* **Step Definition file**
* **Runner file**
* **Data Driven**
* **Scenario Outline**
* **Data Driven Using Examples**
* **Data Driven Using Tables**
* **Tags**
* **Hooks**
* **Tagged Hooks**
* **Execution Order**
* **Background**