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:

. *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.*

. *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.*
. *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.