We discussed how TDD is a test centered development process in which we start writing tests firsts. Initially, these tests fails but as we add more application code these tests pass. This helps us in many ways

We write the application code based on the tests. This gives a test first environment for development and the generated application code turns out to be bug-free.
With each iteration, we write tests and as a result, with each iteration, we get an automated regression pack. This turns out to be very helpful because with every iteration we can be sure that earlier features are working.
These tests serve as documentation of application behavior and reference for future iterations.

Behavior Driven Development

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

. *Shifting from thinking in **"tests"** to thinking in **"behavior"***
. *Collaboration between Business stakeholders, Business Analysts, QA Team and developers*
. *Ubiquitous language, it is easy to describe*
. *Driven by Business Value*
. *Extends Test-Driven Development (TDD) by utilizing natural language that non-technical stakeholders can understand*
. *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 & SpecFlow

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*.

What is SpecFlow?

SpecFlow is inspired by *Cucumber* framework in the Ruby on Rails world. *Cucumber* uses plain English in the Gherkin format to express user stories. Once the user stories and their expectations are written, the Cucumber gem is used to execute those stores. ***SpecFlow brings the same concept to the .NET world*** and allows the developer to express the feature in plain English language. It also allows to write specification in human-readable ***Gherkin format***.

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.

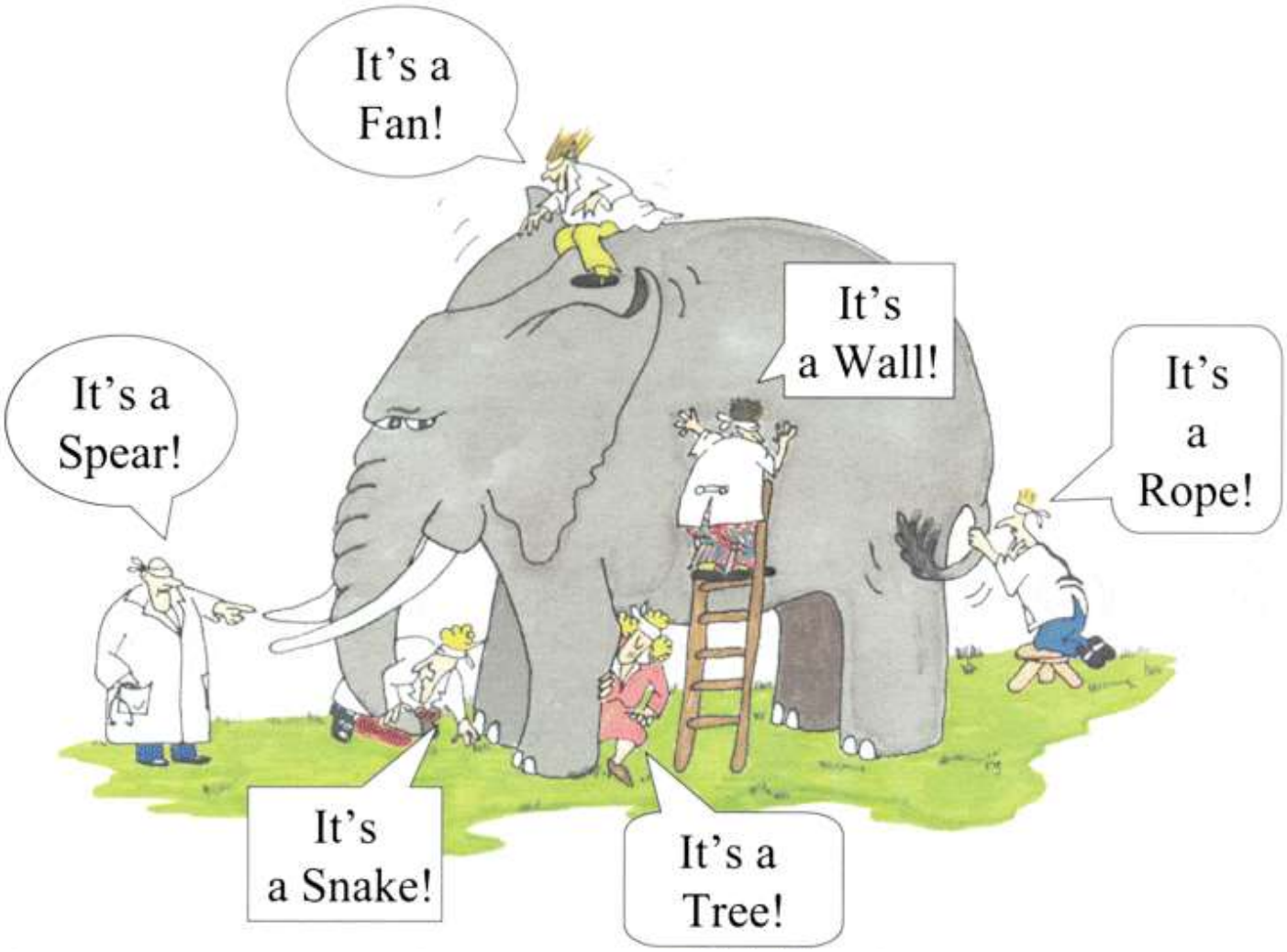
The original idea is awesome. But the only challenge here is that the person who is developing the idea is not the same person who has this idea. If the person who has the idea happens to be a talented software developer, then we might be in luck: the idea could be turned into working software without 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

Where 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.