BDD

Test driven development (TDD)

TDD (test driven development) is software development process in which developers first write the unit tests for feature or module based on requirements and then implement the feature or module itself.

In short TDD cycle is "red > green > refactor":

- ✓ Red – phase where tests are implemented according to requirements, but they still fail
- ✓ Green – phase where module or feature is implemented and tests pass
- ✓ Refactor – phase where working code is made more readable and well structured

Behaviour driven development (BDD)

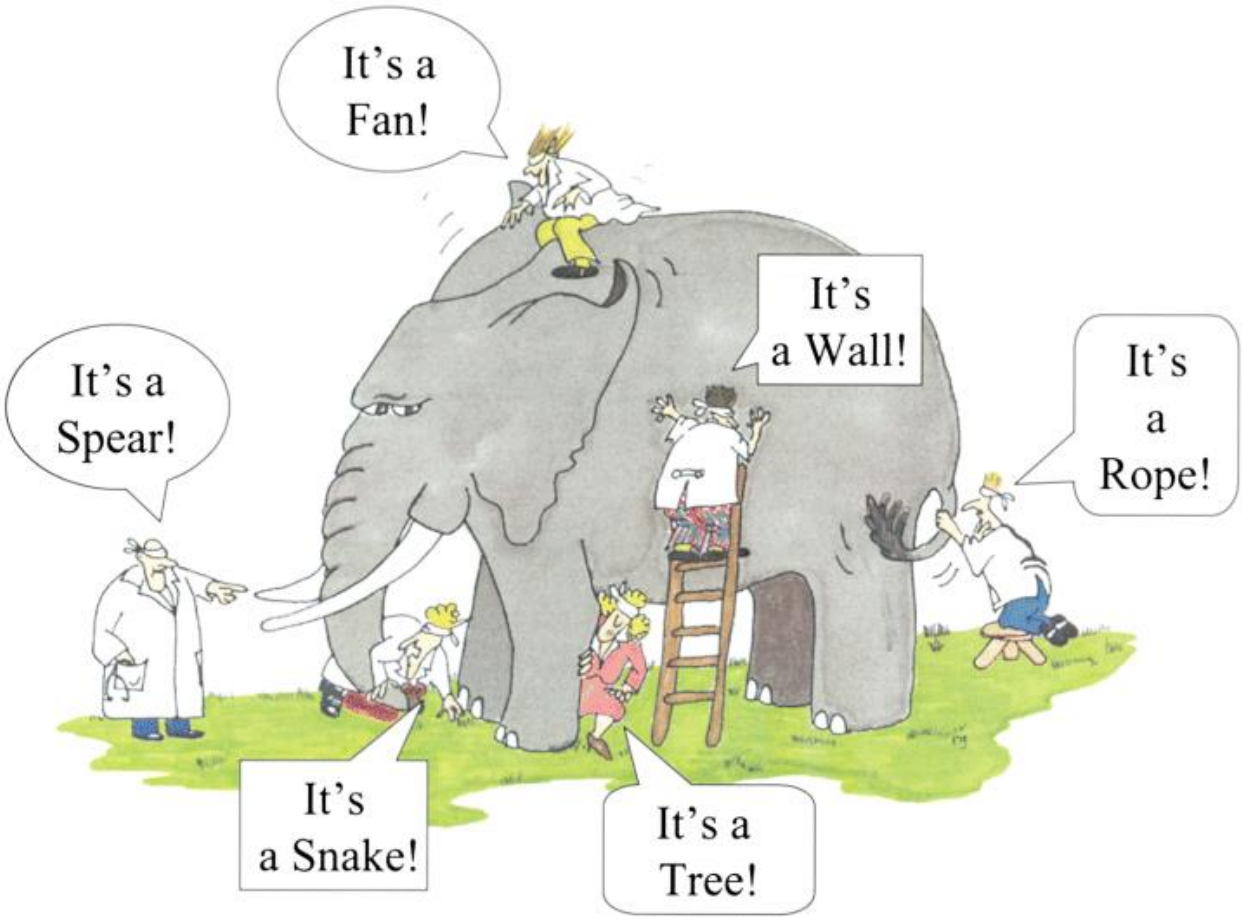
- ✓ BDD emerged from and extends TDD.
- ✓ Instead of writing unit tests from specification why not make the specification a test itself.
- ✓ The main idea is that business analysts, project managers, users or anyone without technical, but with sufficient business knowledge can define tests.

Benefits of TDD

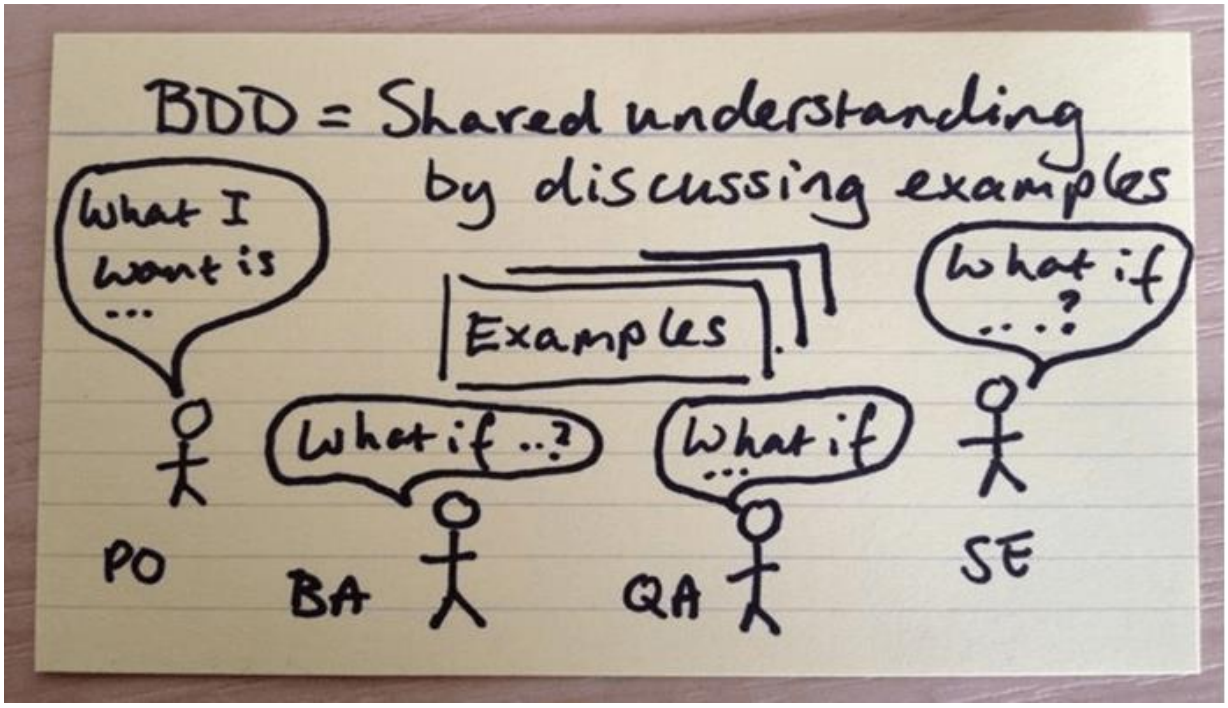
- ✓ Unit test proves that the code actually works
- ✓ Can drive the design of the program
- ✓ Refactoring allow to improve the design of the code
- ✓ Low Level regression test suite
- ✓ Test first reduce the cost of the bugs

Drawbacks of TDD

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


It's a Fan!
It's a Spear!
It's a Wall!
It's a Rope!
It's a Snake!
It's a Tree!

In a nutshell, BDD tries to solve the problem of “understanding requirements with examples”



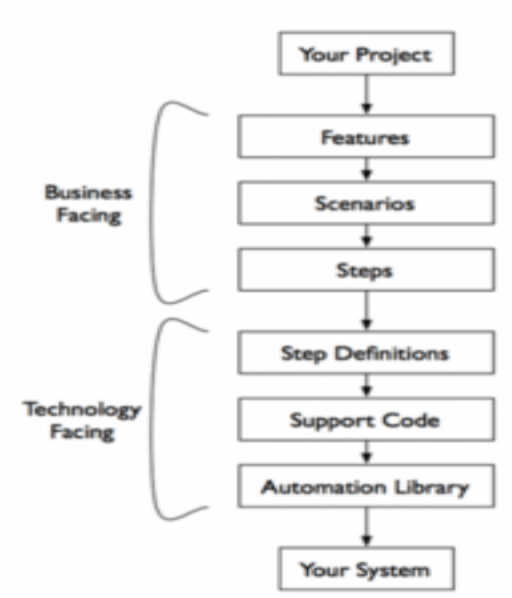
Cucumber:

Cucumber is a test automation tool following the principles of Behavioural Driven Design and living documentation. Specifications are written in a concise human readable form and executed in continuous integration.

Cucumber-JVM

Cucumber is written in Ruby and Cucumber JVM is an implementation of cucumber for the popular JVM languages like Java, Scala, Groovy, Clojure etc

Cucumber Stack



We write features and scenarios in a "Ubiquitous" Language and then implement them with the step definitions and support code.

Feature file and Gherkin

- ✓ You first begin by writing a .feature file.
- ✓ A feature file conventionally starts with the **Feature** keyword followed by **Scenario**. Each scenario consists of multiple steps.
- ✓ Cucumber uses Gherkin for this. Gherkin is a Business Readable, Domain Specific Language that lets you describe software's behavior without detailing how that behavior is implemented.

Example:

```
Feature: Placing bets

Scenario: Place a bet with cash balance

Given I have an account with cash balance of 100

When I place a bet of 10 on "SB_PRE_MATCH"

Then the bet should be placed successfully
```

```
And the remaining balance in my account should be 90
```

As you can see the feature file is more like a spoken language with gherkin **keywords** like Feature, Scenario, Given,When, Then,And ,But, #(for comments).

- ✓ **Step Definitions**
  Once you have finalized the feature file with different scenarios, the next stage is to give life to the scenarios by writing your step definitions.
- ✓ Cucumber uses regular expression to map the steps with the actual step definitions.
- ✓ Step definitions can be written in the JVM language of your choice.
- ✓ The keywords are ignored while mapping the step definitions.
  So in reference to the above example feature we will have to write step definition for all the four steps.

Use the IDE plugin to generate the stub for you.

```
import cucumber.api.java.en.And;

import cucumber.api.java.en.Given;

import cucumber.api.java.en.Then;

import cucumber.api.java.en.When;

public class PlaceBetStepDefs {

 @Given("^I have an account with cash balance of (\\d+) $")

 public void accountWithBalance(int balance) throws Throwable {

 // Write code here that turns the phrase above into concrete actions

 //throw new PendingException();

 }

 @When("^I place a bet of (\\d+) on \"(.*?)\"$")
```

```
public void placeBet(int stake, String product) throws Throwable {

// Write code here that turns the phrase above into concrete actions

// throw new PendingException();

}

@Then("^the bet should be placed successfully$")

public void theBetShouldBePlacedSuccessfully() throws Throwable {

// Write code here that turns the phrase above into concrete actions

//throw new PendingException();

}

@And("^the remaining balance in my account should be (\\d+)$")

public void assertRemainingBalance(int remaining) throws Throwable {

// Write code here that turns the phrase above into concrete actions

//throw new PendingException();

}

}
```

Support Code

- ✓ The next step is to back your step definitions with support code.
- ✓ You can for example do a REST call to execute the step or do a database call or use a web driver like selenium . It is entirely up to the implementation.
- ✓ Once you get the response you can assert it with the results you are expecting or map it to your domain objects.

For example you can you selenium web driver to simulate logging into a site

```
protected WebDriver driver;

@Before("@startbrowser")

public void setup() {

 System.setProperty("webdriver.chrome.driver",
"C:\\devel\\projects\\cucumberworkshop\\chromedriver.exe");

 driver = new ChromeDriver();

}

@Given("^I open google$")

public void I_open_google() throws Throwable {

 driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

 driver.get("https://www.google.co.uk");

}
```

Competitors:

Title	Company	Scope	Application Rights	User Interface Available	Features
FitNesse	GitHub project	Test Automation Framework; Testing Tool	Free Use; Opensource	Batch Mode; Integrated into IDE; Plugin; Web based	
JWebUnit	Sourceforge	Test Automation Framework	Free Use; Opensource	Integrated into IDE; Plugin	
PHPUnit	GitHub project, thePHP.cc	Test Automation Framework	Free Use; Opensource	Command Line; Plugin	
NBehave	GitHub project	Test Automation Framework	Free Use; Opensource	Plugin; Tool Extension	BDD Complex Steps, Binding Code, Built-in Reports, Formatting Flexibility, RegExp Parameterize, Tables Parameterize
UISpec	Google Code Projects	Test Automation Framework	Free Use; Opensource		
Cucumber	Cucumber Ltd	Test Automation Framework	Free Use; Opensource	Ant Integration; Integrated into IDE; Maven Integration; Open CLI; Plugin	Auto Complete, BDD Background Actions, BDD Complex Steps, Binding Code, Built-in Reports, Formatting Flexibility, Hooks, Multi Line Parameterize, RegExp Parameterize, Tables Parameterize, Tagging
Robot Framework	GitHub project, Python	Test Automation Framework;	Free Use; Opensource	COM API; Command Line; Plugin	

Title	Company	Scope	Application Rights	User Interface Available	Features
		Tool Extension			
JBehave	GitHub project	Test Automation Framework	Free Use; Opensource	Ant Integration; Command Line; Integrated into IDE; Maven Integration; Plugin	Auto Complete, BDD Complex Steps, BDD Given Story, Binding Code, Built-in Reports, Hooks, Input External Data, RegExp Parameterize, Steps Prioritization, Tables Parameterize, Tagging
Codeception	GitHub project	Test Automation Framework	Free Use; Opensource	Open CLI; Tool Extension	
SpecFlow	GitHub project, TechTalk	Test Automation Framework; Testing Tool	Free Use; Opensource	Integrated into IDE	Auto Complete, BDD Background Actions, BDD Complex Steps, Binding Code, Built-in Reports, Formatting Flexibility, Hooks, Multi Line Parameterize, RegExp Parameterize, Steps Prioritization, Tables Parameterize, Tagging
Concordion	GitHub project, Google Code Projects	Test Automation Framework	Free Use; Opensource	Command Line; GUI usability; Integrated into IDE; Web based	

Title	Company	Scope	Application Rights	User Interface Available	Features
Frank	GitHub project	Test Automation Framework; Testing Tool	Free Use; Opensource		Record & Playback
Squish	Froglogic	Testing Tool	Commercial; Demo; Trial	Command Line; GUI usability; Integrated into IDE	Custom Libraries, Hybrid test scenario, Record & Playback
Behat	GitHub project	Test Automation Framework; Testing Tool	Free Use; Opensource	Command Line	BDD Background Actions, BDD Complex Steps, Binding Code, Built-in Reports, Formatting Flexibility, Hooks, Multi Line Parameterize, RegExp Parameterize, Tables Parameterize, Tagging
Freshen	GitHub project	Test Automation Framework	Free Use; Opensource	Command Line; Tool Extension	BDD Background Actions, BDD Complex Steps, Binding Code, Built-in Reports, Formatting Flexibility, Hooks, Input External Data, Multi Line Parameterize, RegExp Parameterize, Tables Parameterize, Tagging

Cucumber Jar Link: http://repo1.maven.org/maven2/info/cukes/

Dependency Injection

If your programming language is Java you will be writing glue code (Step Definitions and Hooks) in plain old Java classes.

Cucumber will create a new instance of each of your glue code classes before each Scenario. If all of your glue code classes have an empty constructor you don't need anything else. However, most projects will benefit from a Dependency Injection module to organize your code better and to share state between Step Definitions.

The available Dependency Injection modules are:

- PicoContainer (The recommended one if your app doesn't use another DI container)
- Spring
- Guice
- OpenEJB
- Weld
- Needle

Gherkin:

Cucumber executes your .feature files, and those files contain executable specifications written in a language called Gherkin.

Gherkin is plain-text English (or one of 60+ other languages) with a little extra structure. Gherkin is designed to be easy to learn by non-programmers, yet structured enough to allow concise description of examples to illustrate business rules in most real-world domains.

Annotations:

Annotation is a predefined text, which holds a specific meaning. It lets the compiler/interpreter know, what should be done upon execution. Cucumber has got the following few annotations –

Given –

It describes the pre-requisite for the test to be executed.

Example – GIVEN I am a Facebook user

When –

It defines the trigger point for any test scenario execution.

Example – WHEN I enter "<username>"

Then –

Then holds the expected result for the test to be executed.

Example – THEN login should be successful.

And –

It provides the logical AND condition between any two statements. AND can be used in conjunction with GIVEN, WHEN and THEN statement.

Example – WHEN I enter my "<username>" AND I enter my "<password>"

But –

It signifies logical OR condition between any two statements. OR can be used in conjunction with GIVEN, WHEN and THEN statement.

Example – THEN login should be successful. BUT home page should not be missing.

Scenario –

Details about the scenario under the test needs to be captured after the keyword “Scenario:”

Example –

Scenario:

GIVEN I am a Facebook user

WHEN I enter my

AND I enter my

THEN login should be successful.

BUT home page should not be missing.

Scenario Outline –

Examples –

Background –

Background generally has the instruction on what to setup before each scenario runs. However, it gets executed after "Before" hook (to be covered later). So this is ideal to be used for code when we want to set up the web-browser or we want to establish the database connectivity.

Example –

Background:

Go to Facebook home page.

Comments :

- ✓ Step definition file – If you are using Java as a platform then mark your comments with "//".
- ✓ Feature File – In case of feature file, we just need to put # before beginning your comment.