Behavior Driven Development: An Effective Technical Practice to Develop Good Software

Harshad Naik
Computer & IT Dept.,
VJTI
Mumbai-19, India.

ABSTRACT
Behavior Driven Development (BDD) is the best way to prevent communication gaps within a software development team as well as communication gaps between software developers and stakeholders. This paper talks about what Behavior Driven Development is and why it should be preferred over Test Driven Development. It comprehensively reviews Behavior Driven Development against Test Driven Development. It also talks about implementing Behavior Driven Development using BDD tools such as Cucumber and Selenium and the BDD life cycle.

General Terms
Behavior Driven Development, BDD, TDD, Cucumber, Selenium

Keywords
BDD, TDD, Cucumber, Selenium, Agile.

1. INTRODUCTION
In software engineering, behavior-driven development (BDD) is a software development process that emerged from test-driven development (TDD). Behavior-driven development combines the general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design to provide software development and management teams with shared tools and a shared process to collaborate on software development. In this paper, a review of TDD will be presented first, followed by a comparison with TDD. BDD will be explored in detail.

2. TDD
Test Driven Development requires a person, to write test cases for only the new functionality that is needed, that is not to write test cases for the entire software. Once the person is done writing test cases, he runs the code. Surely all the newly added test cases must fail. If they do fail, then the person starts writing code that will satisfy the given test cases.

In this way test driven development provides a working specification for the code that will be written. It is very widely used by agile developers since it can be used to develop software very quickly.

TDD's biggest advantage is that it assures a developer, that whatever code he is writing will be meaningful, since effectively the developer will write code to satisfy the test cases that were written.

However the biggest drawback with TDD is that the test cases are written in high level languages such as Java and VBScript and there is no way that the stakeholder and Business Analysts can analyze these test cases to see whether they are meaningful. This is where BDD has an upper hand. Since, BDD allows us to write test cases in the form of simple Given, When and Then statements; these can be easily verified by Business Stakeholders or Business Analysts.

3. TDD TOOLS
There are various tools available for Test Driven Development. This paper will talk more about JUnit. JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. JUnit allows us to write test cases in Java, and it can be very easily installed and used in any IDE such as Eclipse or NetBeans. Using JUnit, we write the test cases, run them (ideally they should fail) and then code is written so as to satisfy these test cases. Given below is a simple JUnit test case that was written to test a food ordering application made by me:

```
package admin_business_logic;

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author harshad
 */
public class C_add_itemTest {

    public C_add_itemTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }
```

```
  /**
   * Test of check_unique method, of class C_add_item.
   */
  @Test
  public void testCheck_unique() throws Exception {
     System.out.println("check_unique");
     String dishname = "brownie";
     C_add_item instance =new C_add_item();
     boolean expResult = true;
     boolean result = instance.check_unique(dishname);
     assertEquals(expResult, result);
     // TODO review the generated test code and remove the
default call to fail.
     //fail("The test case is a prototype.");
  }
   @Test
  public void testCheck_unique1() throws Exception {
     System.out.println("check_unique");
     String dishname = "butter chicken";
     C_add_item instance =new C_add_item();
     boolean expResult = false;
     boolean result = instance.check_unique(dishname);
     assertEquals(expResult, result);
     // TODO review the generated test code and remove the
default call to fail.
     //fail("The test case is a prototype.");
  }

}
```

As can be clearly seen, test cases written in JUnit cannot be understood by a stakeholder or business analyst who may not have knowledge about programming languages such as Java.

4. BDD

Behavior Driven Development concentrates on the Behavior of whatever is being built. It is an evolution of Test Driven Development and it allows for better collaboration between the entire team that is developing and testing the software. Feature files allow a stakeholder to write the expected behavior of the software in the form of simple given, when and then statements. This means that Behavior Driven Development ensures that the correct product is being built. That is its biggest advantage. Also being an agile technique, it ensures shorter development times.

5. BDD V/S TDD

Test-Driven Development is a developer practice that involves writing tests before writing the code being tested. Begin by writing a very small test for code that does not yet exist. Run the test, and, naturally, it fails. Now write just enough code to make that test pass. Once the test passes, observe the resulting design, and refactor any duplication you see. Rather than thinking of TDD as a testing practice, it is seen as a technique used to deliver high-quality code to testers, who are responsible for formal testing practices.

And this is where the Test in TDD becomes a problem. Specifically, it is the idea of unit testing that often leads new Test Driven Developers to verify things such as making sure that a register() method stores a Registration in a Registry's registrations collection and that collection is specifically an Array. This sort of detail in a test creates a dependency in the test on the internal structure of the object being tested. This dependency means that if other requirements guide us to change the Array to a Hash, this test will fail, even though the behavior of the object hasn't changed. This brittleness can make test suites much more expensive to maintain and is the primary reason for test suites to become ignored and, ultimately, discarded.

The problem with testing an object's internal structure is that we're testing what an object is instead of what it does. What an object does is significantly more important. The same is true at the application level. Stakeholders don't usually care that data is being persisted in an ANSI-compliant, relational database. They care that it's in the database but even then, they generally mean is that it's stored somewhere and they can get it back [9].

BDD puts the focus on behavior instead of structure, and it does so at every level of development

Once it is acknowledged, it changes the way to think about driving out code. People begin to think more about interactions between people and systems, or between objects, than they do about the structure of the objects. [7]

It is believed that most of the problems that software development teams face are communication problems. BDD aims to help communication by simplifying the language used to describe scenarios in which the software will be used:

Given some context, when some event occurs, then I expect some outcome.

Given, When, Then are simple words that are used whether we are talking about application behavior or object behavior. They are easily understood by business analysts, testers, and developers alike.

6. BDD TOOLS

6.1 Cucumber Introduction

Cucumber supports collaboration and communication between stakeholders and the delivery team. It uses a simple language for describing scenarios that can be written and read with ease by both technical and nontechnical people.

These scenarios represent customer acceptance tests and are used to automate the system we're developing [3].

For every cucumber project there is a single directory at the root of the project named "features". This is where all of the cucumber features will reside. In this directory one will find additional directories, which is step definition and support directories [4].

6.2 Cucumber Feature File

Feature File consist of following components -

Feature: A feature would describe the current test script which has to be executed [6].

Scenario: Scenario describes the steps and expected outcome for a particular test case.

Scenario Outline: Same scenario can be executed for multiple sets of data using scenario outline. The data is provided by a tabular structure separated by (I I).

Given: It specifies the context of the text to be executed. By using data tables "Given", step can also be parameterized.

When: "When" specifies the test action that has to performed

Then: The expected outcome of the test can be represented by "Then".

Consider this sample feature file which is being used to test a sample web page that I created.

Feature: Login Action

Scenario: Successful Login with Valid Credentials

Given User is on login Page

When User enters User Name and Password

Then User should be home page

Feature files form the crux of Behavior Driven Development as it is here that stakeholders or business analysts can specify their business requirements. Basically, Feature files allow us to test the behavior of the software we are building. That is to check whether the product is built correctly and also whether the right product is built.

6.3 Cucumber Test Runner Class

Cucumber uses Junit framework to run. As Cucumber uses Junit we need to have a Test Runner class. This class will use the Junit annotation @RunWith(), which tells JUnit what is the test runner class. It more like a starting point for Junit to start executing your tests. In the src folder create a class called TestRunner. Code sample is shown below:

```
@RunWith(Cucumber.class)
@CucumberOptions(
                features = "sample.feature"
                )
public class TestRunner {


}
```

@RunWith annotation tells JUnit that tests should run using Cucumber class present in 'Cucumber.api.junit' package.

@CucumberOptions annotation tells Cucumber a lot of things like where to look for feature files, what reporting system to use and some other things also. But as of now in the above test we have just told it for the Feature file folder.

6.4 Cucumber Step Definition

To test or run the feature file and in order to test the feature file, it is required to write the implementation or step definition for each step in the feature file in java. When Cucumber executes a Step in a Scenario it will look for a matching Step Definition to execute.

A Step Definition is a small piece of code with a pattern attached to it or in other words a Step Definition is a java method in a class with an annotation above it. An annotation followed by the pattern is used to link the Step Definition to all the matching Steps, and the code is what Cucumber will execute when it sees a Gherkin Step.

For example a few lines from the sample step definition file would be:

```
@When("^User enters UserName and Password$")
public void user_enters_UserName_and_Password() throws Throwable {
driver.findElement(By.id("username")).sendKeys("Harshad");
driver.findElement(By.id("password")).sendKeys("harshad");
driver.findElement(By.id("submit")).click();
}
```

This code fulfills the when statement written in feature file. Here using selenium WebDriver we are passing parameters to the login form.

6.5 Selenium WebDriver

Selenium WebDriver is one of the most important components of Selenium Releases. It is used heavily by the Automation Testing Industry [8].It can be used along with Tools such as Cucumber to carry out Behavior Driven Development. As seen in section 3.4, Selenium WebDriver has been used to fill out a login page.

6.6 Cucumber Selenium Framework

Cucumber and Selenium can together be used to create a framework in which the Cucumber tool mainly serves to write the feature file (Stakeholders requirements can be written here) and Selenium will be used for testing whether the stakeholders requirements were satisfied or not. Together they can be used for Behavior Driven Development. Stakeholders or business analyst will write his requirements in the gherkin format using Cucumber Feature files, then the Developer will develop the software and subsequently testers will test whether the software meets the requirements in the feature files by using Selenium.

7. BDD CYCLE

The BDD delivery cycle starts with a stakeholder discussing a requirement with a business analyst. The requirement might be a problem they want solved or an idea they've had. The analyst helps the stakeholder articulate the requirement in terms of features that make sense to the stakeholder using their own domain terms and maybe further into small, verifiable chunks known as stories, which represent no more than a few day's work. [2]

By identifying which scenarios are important to the story before development starts, the stakeholder can specify exactly how much they want the programmers to do or how much

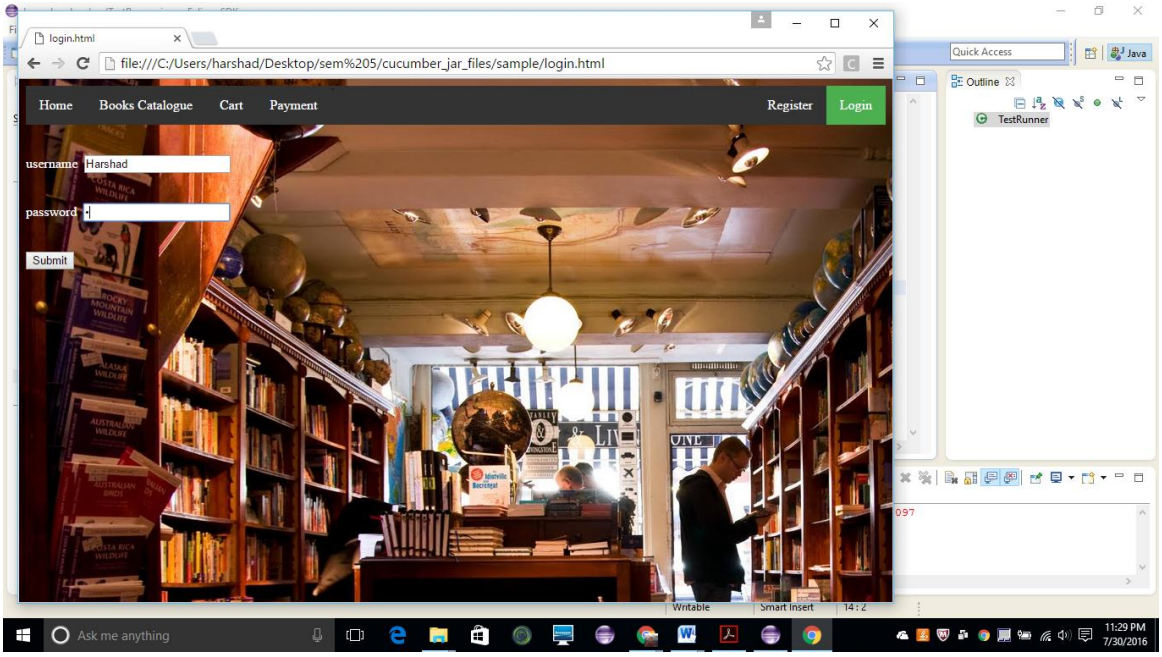

Fig 1: Selenium WebDriver automatically filling in parameters for login form

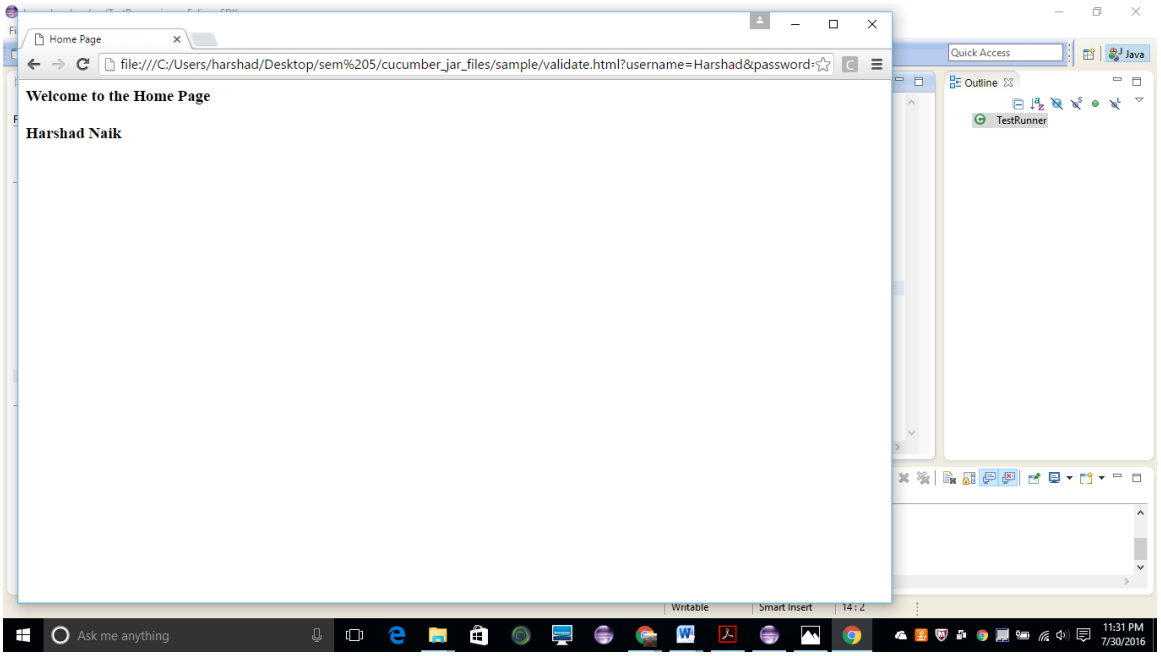

Fig 2: The Home page opened after entering correct login details.

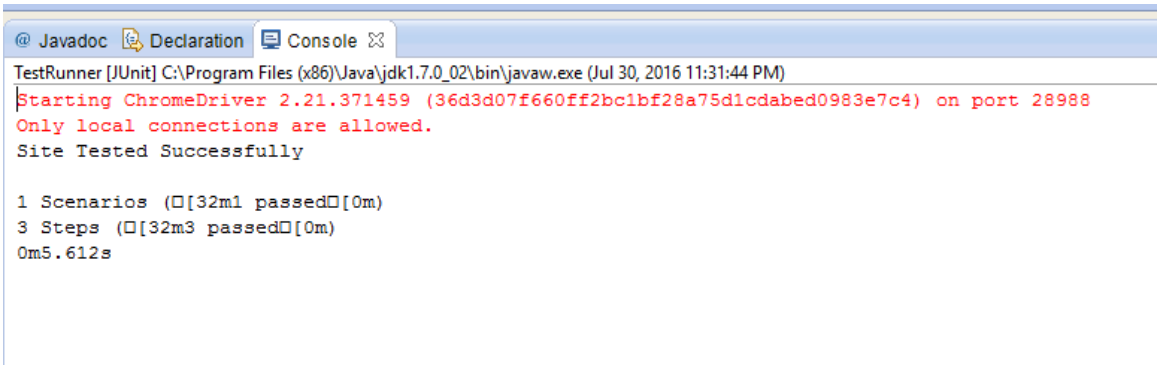

Fig 3 The Java Console tells us that the scenario has run successfully.

Development effort they want to invest in delivering the feature [5]. The developers will only implement enough to satisfy the agreed scenarios, and no more.

The final task before the programmers start implementing the story is to automate the scenarios where it makes sense to do so. In the same way, Test-Driven Development uses code examples to drive the design; these automated scenarios will drive the high-level direction of the development effort.

One of the most important characteristics of BDD is that the scenarios are easy to automate yet are still easily understandable to the stakeholder. Defining and automating these scenarios is the realm of Cucumber. Developers write code to get the scenarios working. Start by writing a code example to describe the behavior needed, then implement the code to make that example work, and then refactor.

Eventually end up with just enough software to make the scenario work, and then iterate through the other scenarios until done. This then brings us full circle, such that it can demonstrate the working scenarios back to the stakeholder, and the story is done.

8. CONCLUSION

In conclusion, BDD is an evolution of TDD, which allows there to be a better collaboration between the members of the software development team. BDD has all the elements of TDD, but it tops it up with feature files which allow one to write test cases in a language that can be understood by all.

Summarily BDD is based on three core principles, namely:

- Enough is enough. Work to achieve the stakeholder's expectations but avoid doing more than needed.
- Deliver stakeholder value. There are multiple stakeholders both core and incidental and everything that is done should be about delivering demonstrable value to them.
- It's all behavior. Just as it can describe the application's behavior from the perspective of the stakeholders, it can describe low-level code behavior from the perspective of other code that uses it [1].

At the start of a project or a release, one can carry out some sort of inception activities to understand the purpose of the work one is doing and to create a shared vision. This is about the deliberate discovery of risks and potential pitfalls along the way.

The day-to-day rhythm of delivery involves decomposing requirements into features and then into stories and scenarios, which we automate to act as a guide to keep us, focused on what we need to deliver. These automated scenarios become acceptance tests to ensure the application does everything that is expected. [7]

BDD stories and scenarios are specifically designed to support this model of working and in particular to be both easy to automate and clearly understandable by their stakeholders [1].

9. REFERENCES

[1] David Chelimsky 'The Rsec Book Behaviour-Driven Development with RSpec, Cucumber, and Friends", 2010.

[2] "Specification by Example" by Gojko Adzic 2011.

[3] "The Cucumber Book" by Matt Wynne and Aslak Hellesoy 2014.

[4] Dave Astels and Steven Baker on RSpec and Behavior-Driven Development

[5] Gojko on BDD: Busting the Myths

[6] Official website of Cucumber https://cucumber.io/.

[7] Blog on BDD https://www.thoughtworks.com/insights/blog/3-misconceptions-about-bdd.

[8] Official website of Selenium http://www.seleniumhq.org/docs/.

[9] Evaluation of Behavior Driven Development by John Horn Lopez 2012.