Behavior-Driven Development

[1]César Duarte, [2]Amílcar Fernandes

Mestrado Integrado Engenharia Informática e Computação

Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL

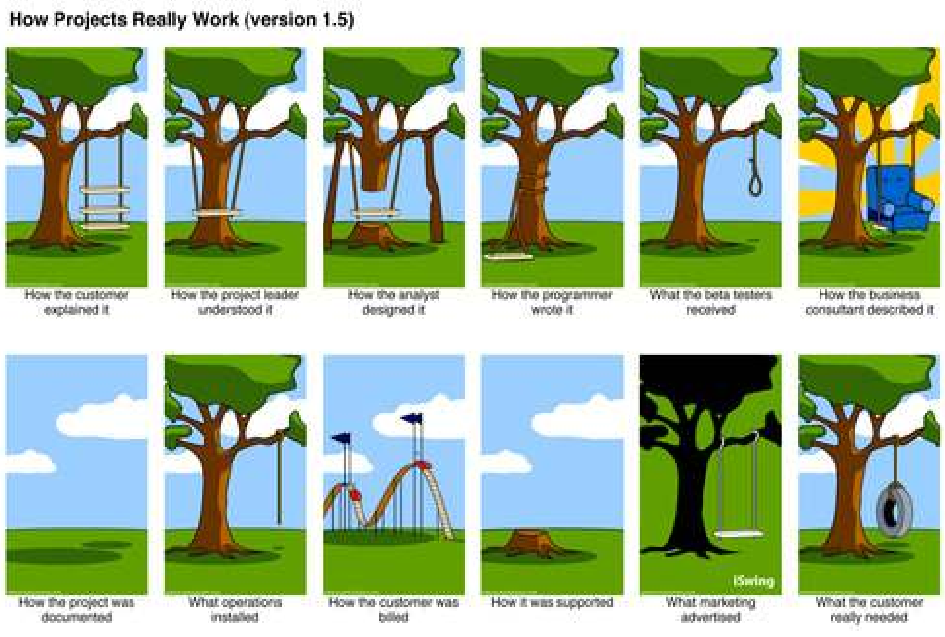
[1]ei06089@fe.up.pt, [2]ei99079@fe.up.pt

Abstract— **This document is intended to be a brief introduction to Behavior-Driven Development (BDD). This methodology is an evolution in the thinking behind Test-Driven Development (TDD) and Acceptance Test-Driven Planning. Its main goal is to help the development team on the delivery of prioritised, verifiable business value by providing a common vocabulary. BDD extends TDD by writing test cases in a natural language that non-programmers can read.**

***Keywords*— agile methodology, software development, collaboration, software behaviour**

I. Introduction

Behavior-Driven Development (BDD from now on) is an agile software development technique that Dan North originally came up with in 2003[1] as a response to Test-Driven Development (TDD from now on) ineffectuality's. It brings together good practices from TDD and Domain-Driven Design (DDD) into an integrated whole. It relies on the use of a very specific (and small) vocabulary to minimise miscommunication and to ensure that everyone – the business, developers, testers, analysts and managers – are not only on the same page but using the same words thus minimizing the hurdles between them and enabling the incremental delivery of business systems. The core of BDD is "Getting the words right", i.e. producing a vocabulary that is accurate, accessible, descriptive and consistent. Concentrating on finding the right words leads us (programmers) to better understand the very close relationship between the stories we use to specify behaviour and the specifications we implement.


How Projects Really Work (version 1.5)
How the customer explained it
How the project leader understood it
How the analyst designed it
How the programmer wrote it
What the beta testers received
How the business consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What marketing advertised
iSwing
What the customer really needed

II. Background

Behavior Driven Development was created by Dan North in 2003. This concept aims at a clear understanding of the software "they" want, involving all stakeholders in the development phases.

Dan North wanted to put his money where is words where and he created the first framework for BDD, using java, which he named JBehave, followed by a similar one using Ruby know by RBehave, this framework was later integrated in RSpec project. David Chelimsky, Aslak Hellesøy and other developers were involved in developing RSpec and in writing the book "The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends". The first story-based framework in RSpec was later replaced by Cucumber mainly developed by Aslak Hellesøy.

In 2008, Chris Matts, who was involved in the first discussions around BDD, came up with the idea of Feature Injection, allowing BDD to cover the analysis space and provide a full treatment of the software lifecycle.

III. Context of BDD

"It's all behavior", or more accurately, it's all about what a system should do. BDD is all about "Getting The Words Right" - when using a consistent vocabulary, much of the traditional disconnect between Business and Technology simply disappears.

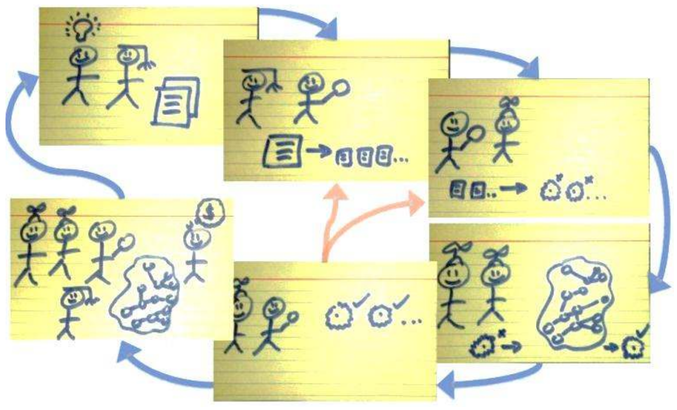
Every behavior of a system should be there because it adds concretely to the business value of the system as a whole.

BDD is driven by business value, that is, the benefit to the business which accrues once the application is in production. The only way in which this benefit can be realized is through the user interface (UI from now on) to the application.

In the same way, each piece of code, starting with the UI, can be considered a stakeholder of the other modules of code which it uses. Each element of code provides some aspect of behavior which, in collaboration with the other elements, provides the application behavior.

The first piece of production code that BDD developers implement is the UI. Developers can then benefit from quick feedback as to whether the UI looks and behaves appropriately. Through code, and using principles of good design and refactoring, developers discover collaborators of the UI and of every unit of code thereafter. By doing this they're acting accordingly to the [1]YAGNI principle, since each piece of production code is required either by the business, or by another piece of code already written.

Fig.2 illustrates the capture of user interface requirements in a given context.



[Fig. 2: BDD user interface]

IV. Implementing BDD

There are a lot of tools to help implementing BDD now days, we will be using JBehave, the first framework made by Dan North, to help us explain how to develop using BDD. JBehave allows us to implement pre-defined BDD concept patterns.

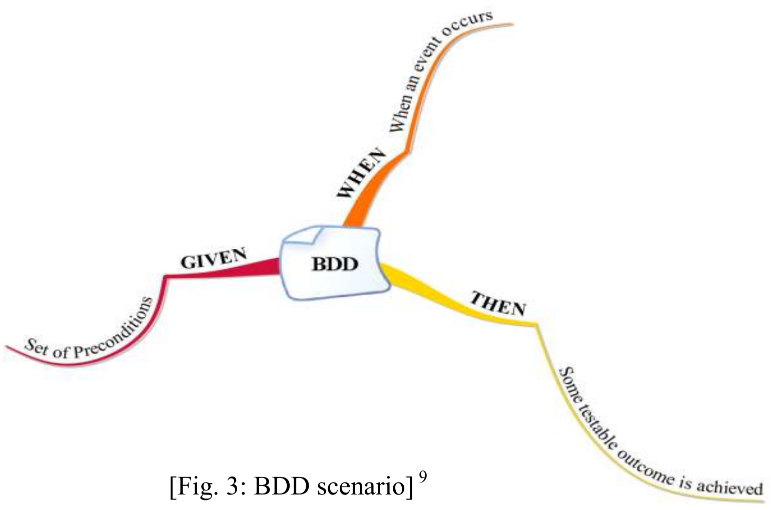
Keeping in mind that BDD is all about behavior, the stories and sceneries must follow a structure, so that BDD tools can understand and interpret them correctly.

Here are how a Story and a Scenario should be define:

1. Story
   |1. As a [role]
   |2. I want [feature]
   |3. So that [benefit]

2. Scenario
   |1. Given some initial context,
   |2. When an event occurs,
   |3. Then ensure some outcomes.

[1]**YAGNI** short for "*You Ain't Gonna Need It"*. Is a principle suggesting that programmers shouldn't add functionality until it is necessary.



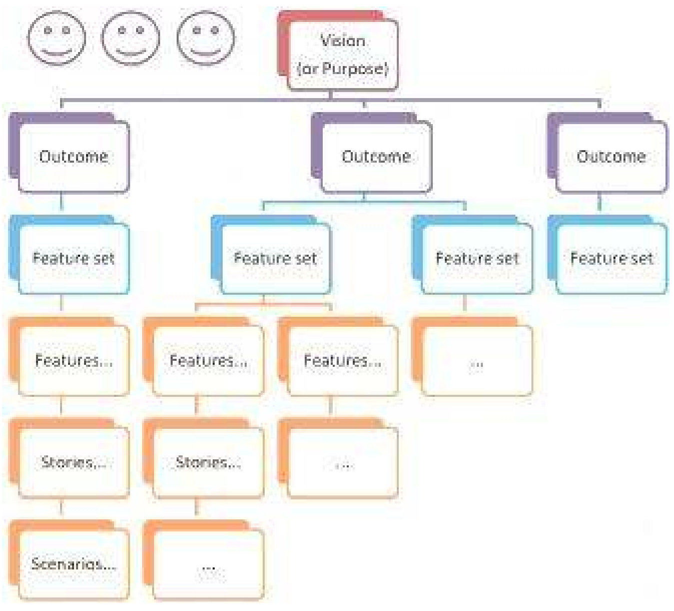
[Fig. 3: BDD scenario] [9]

The acceptance criteria of a Story are presented as scenarios. They are implemented in Java by steps, according to the number of scenarios, which represent story steps. All the classes must inherit JBehave Steps class.

BDD uses a story as the basic unit of functionality, and therefore of delivery. The acceptance criteria are an intrinsic part of the story – in effect they define the scope of its behavior, and give us a shared definition of "done".They are also used as the basis for estimation when we come to do our planning.

Most importantly, the stories are the result of conversations between the project stakeholders, business analysts, testers and developers.

Fig. 4 shows a simple diagram that illustrates the Hierarchy of Stories and Scenarios.



[Fig. 4: Stories vs Scenarios Hierarchy]

V. Process

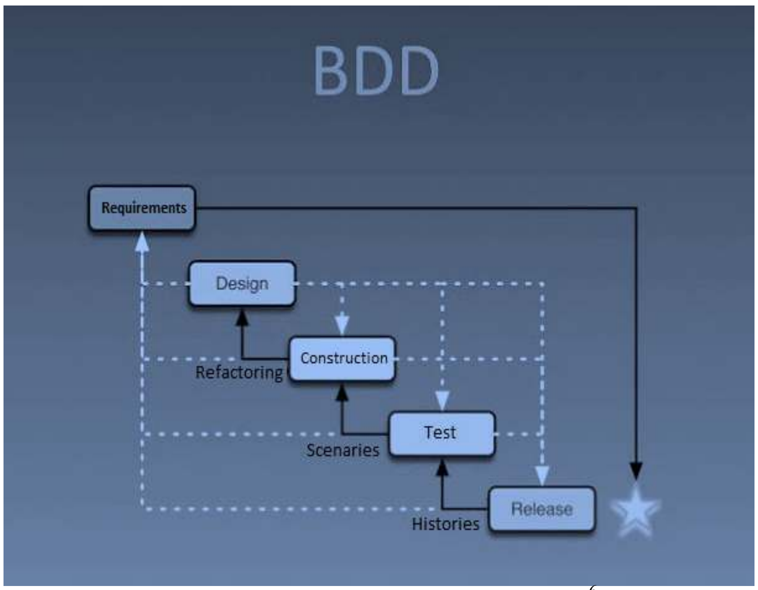
BDD is an agile methodology and it uses pretty much the same process as others agile methodologies, based on iterative and incremental development. In BDD, a developer or QA (Quality Assurance) engineer might clarify the requirements by breaking this down into specific examples (scenarios). Each scenario is designed to illustrate a specific aspect of behavior of the application.

When discussing the scenarios, participants question whether the outcomes described always result from those events occurring in the given context.This can help to uncover further scenarios which clarify the requirements.

The words Given, When and Then are often used to help drive out the scenarios, but are not mandatory.

These scenarios can also be automated, if an appropriate tool exists to allow automation at the UI level. If no such tool exists then it may be possible to automate at the next level in the level of the Controller if an [2]MVC design pattern has been used.

Fig. 5 illustrates the BDD agile process following agile software development processes but with key differences.



[Fig. 5: BDD Process Diagram] [6]

[2] **Model-view-controller** (**MVC**) is a architectural software pattern, it isolates "domain logic" (the application logic for the user) from the UI, permitting independent development, testing and maintenance of each.

VI. Application Example[8]

Easyb is a Groovy based tool. The following example shows how behavior is specified at unit-level (class-level):

```
scenario "Two amounts with the same currencies are added", {
   given "Two different amounts with the same currencies", {
     money1 = new Money(12, "CHF")
     money2 = new Money(14, "CHF")
     expected = new Money(26, "CHF")
   }
   when "Add given amounts" , {
     result = money1.add(money2)
   }
   then "New amount is sum of two given ones", {
     result.equals(expected).shouldBe true

   }
 }

scenario "Two amounts with different currencies are added", {
   given "Two amounts with different currencies", {
     money1 = new Money(12, "CHF")
     money2 = new Money(14, "EURO")
   }
   when "Add given amounts", {
     add = {
       money1.add(money2)
     }
   }
   then "Operation should fail", {
     ensureThrows(IllegalArgumentException) {
       add()
     }
   }
 }
```

Executing the above story gives you the following readable report:

```
   Story: money

     scenario Two amounts with the same currencies are added
       given Two different amounts with the same currencies
       when Add given amounts
       then New amount is sum of two given ones

     scenario Two amounts with different currencies are added
       given Two amounts with different currencies
       when Add given amounts
       then Operation should fail
```

VII. CONCLUSION

The ultimate goal of BDD is “Effective Design and Clean code” this leads to a great deal of advantages, such as tangible stakeholder value, the code being delivered, incrementally, on time, code easy to deploy and manage, robust in production and easy to understand and communicate.

BDD is as much about the interactions between the various people in the project as it is about the outputs of the development process.

It’s an “outside-in” methodology. It starts at the outside by identifying business outcomes, and then drills down into the feature set that will achieve those outcomes. Each feature is captured as a “story”, which defines the scope of the feature along with its acceptance criteria.

While it is likely too new to be considered a "best practice", it must be emphasized that "BDD is a rephrasing of existing good practice."

In conclusion BDD is a technique for improving collaboration on a software development project between the technical (developers, testers, etc) and typically non-technical (management, clients, etc) participants.

It’s a step in the right direction but is up to you to take the other steps to control your project, like deciding what is important. Programming takes time and we don’t usually have a lot of ways to make that faster within a project. But choosing wisely the features that you really need to start “selling your products” (business value) can make the difference between a 4 week and a 6 months project.

REFERENCES

[1] Introducing BDD « DanNorth.net. [Online]. Available: http://blog.dannorth.net/introducing-bdd/

[2] BehaviourDrivenDevelopment, BddWiki [Online]. Available: http://behaviour-driven.org/

[3] In pursuit of code quality: Adventures in behavior-driven development [Online].Available: http://www.ibm.com/developerworks/java/library/j-cq09187/index.html

[4] Human Matters: Why I Started Using Behavior Driven Development [Online].Available: http://humanmatters.tumblr.com/post/393569612/whyistartedusingbehaviordrivendevelopmentatwork

[5] Behavior driven development – Wikipedia, the free encyclopedia [Online].Available: http://en.wikipedia.org/wiki/Behavior_Driven_Development

[6] Imersão BDD on Rails – e-Genial Simplesmente Notável [Online].Available: http://www.egenial.com.br/imersaobddonrails

[7] BDD em Java na prática, com Jbehave « Keep Thinking [Online].Available: http://continuepensando.wordpress.com/2009/10/16/bdd-em-java-na-pratica-com-jbehave/

[8] CLOSED-LOOP: Classifying BDD Tools (Unit-Test Driven vs Acceptence Teste Driven) and a bit of BDD history [Online].Available: http://blog.jonasbandi.net/2010/03/classifying-bdd-tools-unit-test-driven.html

[9] BDD using SpecFlow on ASP.NET MVC Application – CodeProject [Online].Available: http://www.codeproject.com/KB/architecture/BddWithSpecFlow.aspx