Automated Testing Framework Development based on Social Interaction and Communication Principles

Andrei Contan
Department of Automation
Technical University
Cluj-Napoca, Romania
andreicontan@hotmail.com,

Catalin Dehelean
Department of Specialized Foreign Languages
"Babes-Bolyai" University
Cluj-Napoca, Romania
gravedale01@yahoo.com

Liviu Miclea
Department of Automation
Technical University
Cluj-Napoca, Romania
Liviu.Miclea@aut.utcluj.ro

Abstract— **The speed of development of the IT industry as well as the computational power which are increasing exponentially, create great competitiveness in the process of development but also in the launching of software products on the market. Automated testing comes to help with these challenges by trying to increase the speed of development by offering fast feedback and trustworthy quality by means of repeated runs of the implemented tests. This isn't a problem just on a technical level, but also on a social level, especially in the area of communication and understanding the requirements of the client. This work presents the implementation of an automated testing framework which also addresses the social problems. BDD or "Behavior Driven Development" includes an approach which would like to line up the area of client requests to the technical area, offering a uniform platform of collaboration and development. The implementation of this principle is applied in an MVP (Minimum Viable Product) type project which is meant to demonstrate the technical solution which may draw together, both socially and communication wise, the business teams and the technical implementation teams.**

Keywords—testing process; BDD; automated testing; Gherkin language

I. Introduction

In the internet era, the speed of delivering the software applications is the main challenge to the IT companies. If, in the past, a project ran over several years and the phases of a project would have been measured in months, the current projects have to be delivered over a minimum number of months, while the phases of the project are set for weeks or even days.

With such a high frequency of changes, documenting the functionality is becoming a burden because it is invalidated very fast. Under these circumstances, testing the software applications takes place on two levels:

1. The proper development of the product (technically speaking)
2. The development of the proper product (from the point of view of the final user)

In order for the application be successful, both levels must be successful.

In the first phase of this work the working principle of the BDD concept will be presented, to be followed later by a presentation of a generic structure of the automated testing frameworks. Then it will continue with the integration of the two, in order to create the concept of realizing the framework by facilitating the collaboration and communication.

In order to achieve the correct development of the product, the development practices must cover the following requirements:

1. To ensure that the final beneficiary and the development team have a unitary understanding of what needs to be delivered
2. The specifications must be precise for the development team to avoid the useless work caused by ambiguous specifications and functional gaps
3. A common understanding of what a finalized and complete delivery means
4. Documentation to facilitate the changes from the point of view of functionality as well as of team structure.

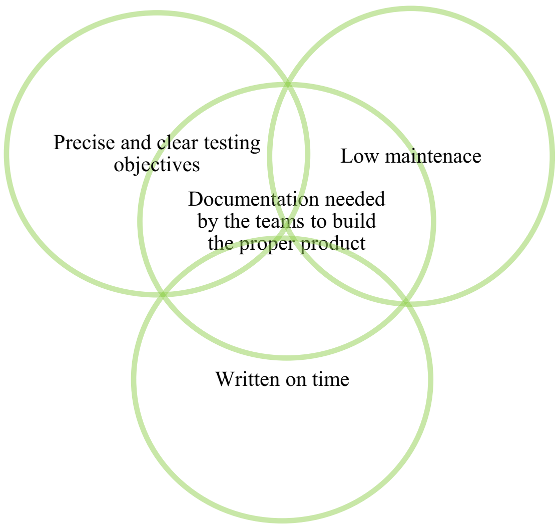
Precise and clear testing objectives
Low maintenace
Documentation needed by the teams to build the proper product
Written on time

Fig. 1.

II. Introduction to BDD

The key to success in BDD lays with the execution of the acceptance tests which describe in an easily understood and easily definable manner a scenario which consists in defining the context, the executed actions and the expected response.

BDD (Behavior Driven Development) – is a growing methodology which relies on the principles of Agile. It was developed by Dan North [1] as an answer to the principles promoted by the TDD (Test Driven Development) approach. BDD uses a level of abstraction which allows for the use of the framework by non-technical people. The level of abstraction consists in a DSL (Domain Specific Language) called the Gherkin language.

In BDD, the scenarios of the acceptance tests are explicitly defined by the following syntax [B1, B6, B7]:

Given <defining an initial context>

When <executing certain actions or steps>

Then <defining the expected results>

Fig. 2.

The requirements of the software applications or the behavior of the application are defined using the former structure, where the keywords, Given, When, Then are defined as Steps.

Each set of steps is parsed and executed by a specific BDD framework which transforms the business requirements into technical details, code classes and test methods.

This scenario format is proving very useful in aligning knowledge and understanding the common principles among programmers, testers and the final client. Moreover, these steps will serve as documentation for the developed application.

III. Other studies in the field of BDD

The research in the field of BDD and its integration in the process of testing leads to a lack of articles and effort invested in this field. The majority of the articles address BDD only as a technique specific to the development of the software applications.

As a result of the analysis, a need to address the social benefits of using this principle, namely BDD, has also been identified.

Carvalho et al [4,5] view BDD as a documentation technique which "automatically validates that the business specifications are properly addressed by the code implemented via the connection mechanisms between the textual description and the technical implementation"

According to the same authors, BDD starts from the idea of the textual description of the requirements using keywords which mark a certain type of sentence and indicate how the that sentence will be translated during the phase of development.

Similarly, Tavares et al. [3] address the implications of the BDD as an integration technique of the process of checking and validating during the design phase of the development process.

Keogh [6] approaches the BDD principles from a larger perspective and questions the value of BDD throughout the entire development process, especially during the interaction between the client and the technical implementation teams.

The author then carries on and claims that BDD is a way which encourages learning, interaction and conversations which may generate new ideas of implementation or testing and even offer improvement feedback.

Even if the study the study in [6] does not provide a detailed list of characteristics of the BDD principle, it does manage to prove that BDD offers far more varied and extensive benefits than the simple TDD differentiation.

IV. The analysis and research approach

Based on the analysis and documentation, this work is meant to answer the following questions:

1. How can an automated testing framework incorporating the BDD principles be developed and / or implemented?
2. Which is the project structure which may facilitate the collaboration and interaction within a team?

Based on the research in [12] and on the list of frameworks described therein, Cucumber[2,7] has been chosen as utility support for the BDD concept. The decision was purely subjective and based on previous experience with this utility and its flexibility to be used in various other technologies

V. Describing functionality as scenarios and templates

A scenario in the BDD format is defined as follows:

Scenario 1: [Title of scenario] **Given:** [Context] **And:** [Extra and optional details **When:** [Action taken] **Then:** [Expected result] **And:** [More details on the result]

Fig. 3. i

A scenario describes how a system which offers any functionality should behave in a certain context, under various actions. The result of this scenario is an action which changes the state of the system or yields an expected result.

In BDD, all scenarios run automatically. The code classes which implement the scenarios, will read the specifications and will execute them. In other words, BDD gives the possibility to have functional executable scenarios.

The mapping rules offer a standard of mapping the scenarios at code level. The name of the file of specifications is mapped to a code class. Each step of the scenario is associated 1:1 to a testing method in the class of code. Every testing method contains an attribute which describes the implemented step.

Cucumber uses regulated expressions to do these mappings. The names of the steps defined in the scenarios should match (by validating with regular expressions) the names of the generated test methods.

Needs and challenges in the development of the automated testing frameworks based on BDD principles:

A. Needs addressed during implementation

1. **Reusing the implementation steps for the BDD scenarios:** testers would like to be able to reuse the implementation of the common elements, identified in defined scenarios. The main reason for reusing is to avoid the fragments of duplicated code for each defined scenario.
2. **Uncoupling the code sequences in business, testing and implementation models:** The final client would like the defined scenarios to be uncoupled from the source code of the automated tests. For the functionality to be implemented, uncoupling is very useful in maintaining collaboration and avoiding code conflicts when several people work on the same component. In its uncoupled form, the clients will concentrate on defining scenarios, testers on test implementation, and the programmers on the implementation of the functionality which forms the bases of the tested system.
3. **Reader and user friendliness:** From a client's perspective, one would like to be able to execute the defined scenarios without access to the code and with no need for technical knowledge to understand the defined implementation. Reporting the results as Pass/Fail has to be done at scenario level to at the test method level.

B. Needs addressed during implementation

1. Maintenance
   a. Framework modeling on integrating a new Cucumber version – implementing the automated testing solutions which are based on dedicated BDD libraries. In this case it's about Cucumber [8]. In the case of a Cucumber library upgrade, the framework must be flexible enough to easily allow such a change.
   b. Refactoring the test code – when changing the already implemented functionalities, refactoring or changing the tests may be a problem of effort and efficiency. Keeping a simple and concise structure in the organization of test scenarios and classes, will allow a refactoring ease and an increased trust in the Endeavour of doing these changes.
   c. Black Box Testing of some corner classes – the BDD principle has some limitations. Since BDD encourages the implementation of end-to-end scenarios, some scenarios, which may be identified in the systems of production, cannot be implemented due to the high level of complexity. Here are some examples thereof: system scalability, login system, monitoring, etc.

Implementing the automated testing solution using the BDD principles

In defining the testing framework, the 3 previously described needs have been taken into account:

i. Reusing the implementation steps;
ii. Uncoupling the sequences in sub-modules;
iii. Ease of reading and use.

The implemented framework consists of defining four modules:

1. Scenario suit steps on using the Gherkin language;
2. Class suite containing test models;
3. Result reporting module;
4. Reusable method package.

1. Scenario suite – represents the collection of scenarios described by the business people in the Gherkin format. The structure of the directory is:

```
resources
│          └── Features
│               ├── ApiFeatureFiles
│               │    ├── CreateStory.feature
│               │    ├── Login.feature
│               │    └── Registration.feature
```

Fig. 4.

The content of a .feature file in the Gherkin language

```
Feature: Create story via API
Scenario: Create story test
Given I send "valid" "POST" call for "login"
Then the response status code is "200"
When I create a new story
Then the response status code is "200"
```

Fig. 5.

1. Class suite containing the test models

```
src/test/java/StepDefinitions/ApiStepDefinitions/
├── ApiSteps.java
├── General.java
├── LoginAndRegistration.java
└── Story.java
```

Fig. 6.

The content of a test class which contains the methods of implementation described in point 1:

```
@Given("^I send a POST call$")
public void i_send_a_POST_call() throws Throwable {
   PostsCast data = new PostsCast();
   data.setAuthor("Author Name");
   data.setComment("This is the day!");
   Response response =
   ApiBaseClass.PostRequest("http://localhost:3000/posts",
   data.toString());
   PostsCast post = response.as(PostsCast.class);}

@Then("^I verify the last added post$")
public void i_verify_the_last_added_post() throws
Throwable {
   Response response =
   ApiBaseClass.GetRequest(apiBaseUrl + "/posts/");
   PostsCast[] post = response.as(PostsCast[].class);
   String expectedStatusCode =
   PostsTestData.getStatusCode();
        Assert.assertEquals(expectedStatusCode, 200);}
```

Fig. 7. k

1. Reporting module for the results of the tests run. Green means that the tests returned the expected result, red signifies a different result from the one expected.

```
When I create a new story
Then the response status code is "200"
```

```
Given I send "invalid" "POST" call for "login"
```

Fig. 8.

1. The package of reusable methods and classes

```
src/test/java/Helpers/
├── ApiBaseClass.java
├── BaseClass.java
├── TestData
│   ├── PostsTestData.java
│   ├── ResponseTestData.java
│   └── TestData.java
└── Utils
    └── Helpers.java
```

Fig. 9.

VI. Future Studies

It is desirable that the next stages of research in the field of BDD and the subsequent implications focus on the different architectural typologies as well as on the social impact within the development teams. It is also desirable to initiate several quantitative and qualitative analysis which are meant to identify the social and communication benefits which the BDD principle promotes and how this principle ultimately reflects on the quality of the delivered software.

VII. Conclusions

The automated acceptance tests are viewed as one of the most efficient ways of creating robust and trustworthy software applications, with a high level of satisfying the expectations of the final clients.

The problem BDD tries to address is the interaction between team members, aligning to the common understanding of the desired needs and functionalities as well as an effective completion of the required defining stages ->implementing->testing->putting into production.

These social principles, integrated into software engineering, reduce the level of ambiguity and the potential conflict areas, while permitting the alignment of unique and clear objectives for all those involved in the development process.

References

[1] D. North, "Introducing BDD," 2006. Available at: http://dannorth.net/introducing-bdd. Accessed on April 3rd, 2016

[2] Cucumber, http://cukes.info/. Accessed on April 3rd, 2016

[3] H.P. Tavares, G. Guimaraes, V. Mota, R. Soares, and R. Atem de Carvalho, "A tool stack for implementing Behaviour-Driven Development in Python Language," CoRR, 2010.

[4] R. Carvalho, F.L. de Carvalho, and R. Soares, "Mapping Business Process Modeling constructs to Behaviour Driven Development Ubiquitous Language," CoRR, 2010

[5] R. Carvalho, F.L. De Carvalho, and R. Soares, "Mapping Business Process Modeling constructs to Behavior Driven Development Ubiquitous Language," CoRR, 2010.

[6] E. Keogh, BDD: A Lean Toolkit. In "Processings of Lean Software & Systems Conference", Atlanta, 2010.

[7] D. Chelimsky, D. Astels, Z. Dennis, A. Hellesoy, and D. North, "The RSpec book: Behaviour Driven Development with RSpec, cucumber and friends," Pragmatic Bookshelf, 2010.

[8] D. Janzen, D.H. Saiedian, "Test-driven development: concepts, taxonomy, and future directions," Computer, vol.38, no. 9, pp. 43-50, Sept 2005.

[9] D. North, "Introducing behaviour driven development," 2006. Available online at: http://dannorth.net/introducing-bdd/. Accessed on April 3rd, 2016

[10] Gherkin. Available online at: https://github.com/cucumber/tree/master/gherkin, Retrieved on April 1st, 2017.

[11] Acceptance Testing. Available online at: http://guide.agilealliance.org/guide/acceptance.html, Retrieved on Nov 1, 2014.

[12] C. Solis, X. Wang, "A Study of the Characteristics of Behaviour Driven Development," 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011.