Applying Executable Specifications (A Short Project Summary & Survey)

Reuven Yagel and Ori Sarig

Software Engineering Department, The Jerusalem College of Engineering
POB 3566, Jerusalem, 91035, Israel

Abstract. This paper summarizes a student graduation project concerning applying executable specifications. It reviews some of the advantages of executable specifications and how it could help the communication between the developer, the software tester and the client. It draws some conclusions trying to explain why in spite of their great potential, they are still not an essential part of the common software development toolbox today.

1 Introduction

*Executable specification*s are written specifications that can be executed by a machine in order to validate the implementation of a system [1, 18]. In a previous paper [18] we claimed that these have great potential for the success of software development projects. This paper draws some practical conclusions which where learned while applying such methods during a student graduation project. We start briefly with some benefits of practicing executable specifications for software development. Then we report on the actual experience gained from a project using it. We conclude with conclusions we learned while practicing it.

2 Executable Specifications – Benefits and Variants

Using such methods can help bridge the known gap between the requirements of a product and an implementation [18, 2]. This is since the encouraged process is to write them with various degrees of involvements of both business facing stakeholders together with more technical/engineering facing persons. In contrast to document based specification, executable specifications can be verified constantly by a machine (even as part of a continuous integration and deployment settings [7]). Thus shortening the feedback loop and allowing agile development that fits better to the customer's real needs. If done correctly the executable specifications stays relevant during the product lifecycle and become a living de-facto documentation of what the system should do and actually is doing!

It seems that theses and related practices are still being developed and defined these days. For completeness of the discussion we briefly quote here from our previous paper [18] in which a review of some variants in this area is given (see there

Yagel R. and Sarig O..
Applying Executable Specifications - (A Short Project Summary & Survey).
DOI: 10.5220/0004179800670070
In *Proceedings of the 3rd International Workshop on Software Knowledge* (SKY-2012), pages 67-70
ISBN: 978-989-8565-32-7

also for some tooling review). Many of these methods emerged out of the test first community [6]. The first one is known as Acceptance Test Driven Devolvement (ATDD) or many times just Agile Acceptance Testing (e.g. [9]). This practice arose as an extension to the unit-testing practice of Test Driven Development (TDD) [3]. Instead of specifying in code only the interface and required behavior of specific modules, these methods extend into developing a set of scripts which demonstrate the various behaviors of the system. Since the execution of those scripts can be automated, it is sometimes also called automated functional testing.

Another related method is Behavior Driven Development (BDD) [10]. This is also a practice (or a group of methods) augmenting TDD with emphasis on stakeholder readability and shared understanding.

The agile software development community keeps evolving these methods and recent representatives and suggested names are, e.g., Story Testing, Specification with/by examples [2] and lately just Living/Executable Documentation (e.g. [4]). Further detailed reviews can be found, e.g., at [1, 2].

3 Project Description

The described project started by the student learning the discussed methods and tools. In parallel, another student was starting her project and the idea was that the first student will help gather and specify the other project's requirements by means of executable specifications.

Since the selected project was being implemented as a website using the Microsoft ASP.NET MVC platform [14], the student started researching and practicing the related tools available in this platform and especially with a BDD tool called SpecFlow [15].

Here it is interesting to note that for various reasons these two parallel efforts did not converge well. So we shifted the goal of the described project to write acceptance test for an ongoing project using the acquired skills. This time it worked better and the result was a comprehensive test suite using the following tools:

- Gerkin [5] for writing the executable specifications.
- SpecFlow [15] for translating into runnable C# code.
- Watin [16] for web automation of the developed product.
- Nunit [12] for running the specifications as tests.
- Jenkins [8] for continuous and automatic execution of the tests upon code changes.

Figure 1 shows one example from those specifications. The target project reported that it got a few benefits from the developed tests: overall it became much more consistent (e.g., between functionalities scattered in various site pages). The specifications became (although after the fact) a basis for describing the product (but not really a useful specification). Finally a few regression bugs were caught and corrected quickly due to the existence of the specifications as acceptance tests.

```
[When(@"I mark the Airport checkbox")]
public void WhenIMarkTheAirportCheckbox()
{
    CheckBox checkbox = browser.CheckBox(Find.ByName("[0].CheckedColumn"));
    checkbox.Click();
}
[When(@"I choose Equals from the combobox")]
public void WhenIChooseEqualsFromTheCombobox()
{
    SelectList sl1 = browser.SelectList(Find.ByName("[0].Operator"));
    sl1.Option("שווה ל-").Select();
}
[When(@"I type Eilat in the textbox")]
public void WhenITypeEilatInTheTextbox()
{
    TextField textfield = browser.TextField(Find.ByName("[0].ColumnValue"));
    textfield.TypeText("אילת");
}
[When(@"I Press on the above database information button")]
public void WhenIPressOnTheAboveDatabaseInformationButton()
{
    browser.Button(Find.ByValue("קבל מאגר >>")).Click();
}
[Then(@"I should see Eilat Airport Table")]
public void ThenIShouldSeeEilatAirportTable()
{
    Assert.IsTrue(browser.ContainsText("אילת"));
}
```

Fig. 1. A specification example.

4 Conclusions

Currently it seems that the Ruby and Rails communities are leading in using and benefiting from executable specifications, e.g., [5, 17]. In order to be accepted widely, the industrial strength platforms (e.g. Java and .Net) needs tools; in at least the same degree of matureness. The student found that: "it is not that hard to find tools, but it is hard to find examples and tips about these tools online".

Additionally, the synergy between the tools is very important here, especially in those environments. The .Net community is showing signs for improvements, e.g., by means of the NuGet [11] package manager which most of the relevant and related tools are already supporting.

At the end, the main contribution of using executable specification in this project was actually to testing. So the main claim that it should help improving communication between different stakeholders was not really validated. It also seems that historically the discussed methods were developed first as testing methods. So the question remains whether they are widely applicable also to better specifications (see also [13] for a recent and detailed discussion about the connection between TDD and BDD).

At last, naturally the development of the specification/tests has leaded the specification writer to become familiar with the domain knowledge of the target project. It seems that this can become an iterative process in which the domain knowledge and the implementation are being explored and developed together in a

way that can bring for optimum elegance, simplicity and correctness of the software product.

We would like to thank the SKY 2012 reviewers for their very helpful comments.

References

1. Adzic, G., Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri, London, UK, 2009.
2. Adzic, G., Specification by Example – How Successful Teams Deliver the Right Software, Manning, New York, USA, 2011.
3. Beck, K., Test Driven Development: By Example, Addison-Wesley, Boston, MA, USA, 2002.
4. Brown, K., Taking executable specs to the next level: Executable Documentation, Blog post, (see: http://keithbrown42.wordpress.com/2011/06/26/taking-executable-specs-to-the-next-level-executable-documentation/), 2011.
5. Chelimsky, D., Astels, D., Dennis, Z., Hellesoy, A., Helmkamp, B., and North, D.: The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends, Pragmatic Programmer, New York, USA, 2010.
6. Freeman, S., and Pryce N.: Growing Object-Oriented Software, Guided by Tests, Addison-Wesley, Boston, MA, USA, 2009.
7. Humble J., and Farley D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, Boston, MA, USA, 2010.
8. Jenkins: (see: http://jenkins-ci.org/), 2012
9. Koskela L., Test Driven, Practical TDD and Acceptance TDD for Java Developers, Manning Publications, 2007.
10. North, D.: "Introducing Behaviour Driven Development", Better Software Magazine, (see http://dannorth.net/introducing-bdd/), 2006.
11. NuGet (see: http://nuget.codeplex.com/), 2012.
12. NUnit: (see http://www.nunit.org), 2012.
13. Pais M. Virtual Panel: Code-to-Test Ratios, TDD and BDD, (see: http://www.infoq.com/articles/virtual-panel-tdd-bdd), 2012
14. Sanderson S., Pro ASP.NET MVC Framework, Apress, 2009.
15. SpecFlow – Pragmatic BDD for .NET: (see http://specflow.org), 2010.
16. Watir, Automated testing that doesn't hurt, (see: http://watir.com/), 2012.
17. Wynne, M. and Hellesoy, A.: The Cucumber Book: Behaviour Driven Development for Testers and Developers, Pragmatic Programmer, New York, USA, 2012.
18. Yagel, R.: "Can Executable Specifications Close the Gap between Software Requirements and Implementation?", pp. 87-91, in Exman, I., Llorens, J. and Fraga, A. (eds.), Proc. SKY'2011 Int. Workshop on Software Engineering, SciTePress, Portugal, 2011.