Can Executable Specifications Close the Gap between Software Requirements and Implementation?

Reuven Yagel

The Jerusalem College of Engineering, Software Engineering Department
P.O.B. 3566, 91035, Jerusalem, Israel

Abstract. This paper reviews some of the techniques which are considered executable specifications. It explains why, in spite of their apparent benefit, they did not become an industry mainstream (yet) and what might be done about it. At last several future directions, research plans and suggestions are drawn.

1 Introduction

1.1 Specifications are Hard

“Walking on water and developing software from a specification are easy if both are frozen.” This quote from E. V. Berard [5] demonstrates maybe the biggest challenge of software projects. Today for most software development areas (especially web development) it is quite clear that software must be developed in a "soft" way. This allows the customer or stakeholder to change her mind as the software is being developed.

In contrast, the most common used form of a specification till recently, is a document, e.g., a Software Requirement Specification. For many software projects, writing such a document is a pre-requisite to starting subsequent phases like design, implementation, testing, deployment, etc.

The problem with such documents is that they are relatively hard to update as a software project encounter changes. This situation leads many times to non-relevant specifications and thereafter difficulties in verifying that the correct software was actually built.

1.2 Importance of the Requirement Phase

One might try to minimize the effect of specifications on development by eliminating or at least reducing this phase. This is not realistic for many reasons, two of the majors are: a) quoting F. Brooks famous words: “The hardest single part of building a software system is deciding precisely what to build” [7], which reminds us that there is no way around this phase. b) using data from the known (but also controversial) 1994 Standish Group "Chaos Report" which found that among 8000 projects the

Yagel R..
Can Executable Specifications Close the Gap between Software Requirements and Implementation?.
DOI: 10.5220/0003698800870091
In *Proceedings of the 2nd International Workshop on Software Knowledge* (SKY-2011), pages 87-91
ISBN: 978-989-8425-82-9

number one reason for project failure was lack of user involvement [19], meaning software engineers must understand what the user needs before building it for her (later reports still contain this finding in various ranks).

1.3 Executable Specifications

Recent ideas, especially in the agile development community, try to solve the specification problem by forming requirements in a way that is both readable by client but also can be executed by a machine in order to be able validate the requirements. Thus having a shared document which is both understandable by the user (or business expert) and the development team which can execute it – hoping that in this form the document will stay relevant along the whole project life and especially allowing continuous feedback and validation to the software product status.

For the following discussion *executable specification* will be defined as a written specification that can be executed by a machine in order to validate the implementation.

In the reminder, this paper reviews some main methods and tools in this area, discusses their problems and why the author thinks they did not catch yet and then some suggestions and future directions.

2 Review

2.1 Executable Specification Variants

First, several related practices are identified here, further detailed reviews can be found, e.g., at [2, 3]. These practices are also supported by various tools; the prominent ones are briefly mentioned. Note that these practices are not at all equivalent one to another, but they share the major points discussed here.

Many of these methods emerged out of the test first community [11]. The first one is known as Acceptance Test Driven Devolvement (ATDD) or many times just Agile Acceptance Testing (e.g. [15]). This practice arose as an extension to the unit-testing practice of Test Driven Development (TDD) [4]. 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) [17]. 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 examples [3] and lately just Living/Executable Documentation (e.g. [14]).

2.2 Tooling

Some related tools are quickly reviewed here.

FitNesse [1] is a wiki-based web tool which allows non-developers to write acceptance tests, in a formatted manner, e.g. tabular example/test data. It is built on an earlier tool: Framework for Integrated Test (FIT) [16]. FIT pioneered the possibility of non-programmers to supply tests in the form of Html tables.

Cucumber [8] is a newer tool directly supporting BDD. It is an evolution of previous tools like RSpec Story Runner and RBehave [8]. Their main feature is the ability to run stories written in plain human language (originally English but by now a few dozen others). This tool is also highly connected to unit testing tools and user/web automation tools, discussed next.

Related and relevant tools were developed in recent years, mainly to automate the testing of web applications. Since they mimic user interactions with a web page, they can be used as tools for verifying the whole behavior of (web) applications, effectively reaching acceptance testing. Major representatives of these tools are Selenium [18] and Watir/Watin [20] – but many other exist.

3 The Reality – Slow Adoption

In spite of the apparent benefits of the methods and tools discussed so far, they are hardly used in industry (e.g. [9,10]).

Several alternative explanations are suggested:

1. These methods are relatively new. Some claim that it takes 10-15 years before mainstream industry adapts new ideas [6]. Moreover various surveys show growing adoption of TDD [4], so even though the method discussed here are not so new – the executable specification community hopes to break through.
2. It might be that the tooling suggested so far are not mature enough. Many of them are based on the Ruby language which is more common in a hacking culture rather than at mainstream enterprise development. However in recent years those tools were ported to the main platforms (e.g., Java/.Net), so it seems it is not only the tools.
3. Lastly, I believe that the slow adoption is due to the current emphasis on tooling and developer practices rather than on sharing and openness towards the non-developers involved in the development process. This is a point for further research and possibly suggesting various ways for better communication patterns between the involved parties.

4 Suggestions and Future Research

This short review is due to further investigating of the applicability of executable specification methods. Especially by using existing data related to projects using these methods [3] and also by measuring and comparing their effect on new industrial and/or student projects.

Ontologies [12] and other information/knowledge science ideas can further bring executable specifications to the business/customer world.

Lastly, it seems that over the years, the community is becoming better and better at specifying the scope of software projects. However, this comes with losing the user/stakeholder's true goals. This work would like to help forward the research and practice toward closing this gap.

Acknowledgements

I would like to thank the SKY reviewers for their helpful comments.

References

1. G. Adzic. Test Driven .NET Development with FitNesse, Neuri, 2008.
2. G. Adzic. Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri, 2009.
3. G. Adzic. Specification by Example - How Successful Teams Deliver the Right Software, Manning, 2011.
4. K. Beck. Test Driven Development: By Example, Addison-Wesley, 2002.
5. E. V. Berard. Essays on Object-Oriented Software Engineering, Prentice Hall, 1992 (see: http://www.softwarequotes.com/showquotes.aspx?id=613).
6. M. Bria. "Jim Shore Suggests Automated Acceptance Tests Are Not The Right Move". Infoq article, 2010 (see: http://www.infoq.com/news/2010/04/dont-automate-acceptance-tests).
7. F. Brooks. The Mythical Man-Month Essays on Software Engineering, Addison-Wesley, Anniversary 2nd edition, 1995.
8. D. Chelimsky, D. Astels, Z. Dennis, A. Hellesøy, B. Helmkamp, and Dan North. The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber, and Friends, Pragmatic Programmer, 2010.
9. J. Coplien and B. Martin. "Coplien and Martin Debate TDD, CDD and Professionalism", infoq interview, 2008. (see: http://www.infoq.com/interviews/coplien-martin-tdd).
10. A. Elssamadisy. "Automated Acceptance Tests - Theoretical or Practical", infoq article, 2009, (see: http://www.infoq.com/news/2009/06/automated-acceptance-tests).
11. S. Freeman and N. Pryce. Growing Object-Oriented Software, Guided by Tests, Addison-Wesley, 2009.
12. Gartner, Predicts 2010: Agile and Cloud Impact Application Development Directions, Gartner, 2009.
13. T. R. Gruber. "Toward principles for the design of ontologies used for knowledge sharing", Padua workshop on Formal Ontology, 1993, and International Journal of Human-Computer Studies, Vol. 43, Issues 4-5, 1995.
14. Keithps. "Taking executable specs to the next level: Executable Documentation", Blog post, 2011, (see: http://keithps.wordpress.com/2011/06/26/taking-executable-specs-to-the-next-level-executable-documentation/).
15. L. Koskela, Test Driven, Practical TDD and Acceptance TDD for Java Developers, Manning Publications, 2007.
16. R Mugridge & W Cunningham. Fit for Developing Software: Framework for Integrated Tests, Prentice Hall, 2005.

17. D. North. Introducing Behaviour Driven Development, (see: http://dannorth.net/introducing-bdd/).
18. Selenium. SeleniumHW Web Application Test System. http://seleniumhq.org/
19. Standish Group, "CHAOS Report.", Standish Group, 1994.
20. Watir, Automated testing that doesn't hurt, http://watir.com/.