A Literature Review on Story Test Driven Development

Shelly Park, Frank Maurer
Department of Computer Science
University of Calgary
2500 Univeresity Dr. NW, Calgary, AB, Canada
{sshpark, fmaurer}@ucalgary.ca

Abstract. This paper presents a literature review on story-test driven development. Our findings suggest that there are many lessons learned papers that provide anecdotal evidence about the benefits and issues related to the story test driven development. We categorized these findings into seven themes: cost, time, people, code design, testing tools, what to test and test automation. We analyzed research papers on story test driven development to find out how many of these anecdotal findings were critically examined by researchers and analyzed the gaps in between. The analysis can be used by researchers as a ground for further empirical investigation.

Keywords: Story Test Driven Development, Executable Acceptance Test Driven Development, Requirements, Systematic Review, Testing, Empirical software engineering, Agile software development

1 Introduction

The Story Test Driven Development (STDD) is a way of communicating requirements using tests. The purpose of STDD is to facilitate better communication between the customers and the development team by reducing the ambiguities in the requirements. The testable requirements can either pass or fail, thus story tests reduce the ambiguities in requirements interpretations. This idea is currently called many names in the agile software engineering community: functional tests [1], customer tests [1], specification by example [2] and scenario tests [3], executable acceptance tests [4, 5] and behavior driven development [6] among many.

The idea of STDD has been in circulation in the agile software engineering community for a decade now starting with Beck's publication of his book [1] in 1999. For the last decade, we have seen the industry accept and practice many of the agile concepts in varying degrees despite their initial objections. For example, TDD has been widely accepted by developers in industry. However, comparably STDD is much less adopted and there is still a lot of confusion about what STDD is and where it should be used. The aim of the paper is to analyze the state of our knowledge on STDD. We want to categorize what people found to be the difficult points in practicing STDD and what research has discovered so far in our understanding of STDD. We performed a literature review on STDD papers published in conferences, magazines and journals. We categorized them into lessons learned/experience papers, tool development papers and empirical research papers.

2. Result

This section describes in detail on STDD that are published in various publication venues in the past. We categorized the discussion points into 7 themes: cost, time, people, code design, testing tools, what to test, and test automation issues. We first describe the points from the lessons learned and tool development papers and describe the findings from the research papers. For references that start with [T], please refer to [7].

2.1 Cost

Budget is an important aspect of software development projects, especially when one needs to justify the cost of introducing a new process such as STDD into a development team. Authors in [T15, T16, T17] suggested that the benefit of STDD is to help keep the project within budget. Schwartz also states that the automated story tests can “run often and facilitate regression testing at low cost”[T16]. However, four papers [T15, T16, T18, T19,] stated that STDD may not pay off because the cost of writing and maintaining the tests is high. In addition, four papers stated that their teams did not have the budget necessary to automate the tests [T18, T19, T20, T21]. There were no research papers that explicitly analyzed the cost and budget aspect of STDD process or the STDD tools.

2.2 Time

Five points were discussed in the lessons learned papers as the benefits of STDD process: 1) The STDD can help check the overall progress [T15, T17, T18, T19, T22, T23, T24, T25, T26, T27, T28, T29, T68]; 2) adapt to requirements changes with the help of instant feedback, which can help keep the project on time [T18, T27, T30]; 3) continuous verification (test often, repeatedly)[T17, T25, T27]; 4) better estimation of the stories [T23, T31]; 5) immediate defect fixes [T28, T32]. However, some lessons learned papers identified three issues related to time: 1) writing and maintaining tests took considerable time [T17, T28, T29, T33, T34, T35], 2) it can take long time to execute the tests [T24, 34, 36, 37, 38], and 3) there can be a lack of time to build the necessary testing tools and infrastructures [T28].

There were two research papers that dealt with time on STDD. The research paper, [T72], discovered that the test subjects were able to write and test using story tests within an expected amount of time. The research paper, [T74], discovered that timing was a matter of discipline more than an actual timing problem.

2.3 People

Software is developed by people. Their commitment, skills and collaboration are important in the success of the development project. The lessons learned papers suggest there are five benefits: 1) better communication with the stakeholders [T17, T25, T31, T39, T40, T41, T42, T43, T63, T64], 2) confidence about the progress and deliverables [T17, T19, T28, T32, T38, T39, T41, T44, T45], 3) better awareness for testing in the team [T81, T35, T61, T17, T28, T68], 4) encouragement of collaboration between right people [T39] and 5) anyone can quickly understand what has been developed [T34, T35]. The lessons learned papers also identified two problems related to people. 1) The STDD affects everyone, which made the adoption difficult [T24]. 2) Some papers identified that there was no direct contact between developers and customers because the tests were too explicit [T34, T41]. In addition, there were some papers that discussed the people's skills. Some papers argued that it took too long to learn the STDD testing tool or the specification language [T27, T46, T55]. Some authors identified that lack of test automation experience in the team was the barrier [T25, T43, T46, T48, T69], but most of them overcame the problem quickly.

In terms of the responsibility of writing and maintaining the story tests, there were teams where the whole team was equally responsible for the tests [T19, T26, T35, T48], or a separate group of dedicated developers/testers were created for STDD [T18, 34]. One team used pair story testing method [T18]. In terms of who writes the tests, there were many variations. Some stated that the customers wrote the tests with the help of the developers and testers [T23, T24, T25, T27, T30, T36, T42, T49, T50]. In some cases, developers wrote the story tests with the customer collaboration [T20, T26, T27, T51]. In some teams, the QAs wrote the tests in collaboration with the customer [T19, T45].

We found seven research papers that looked into people related issues. [T67] performed an experiment on how quickly developers can learn to use a STDD tool and discovered that 90% of the test subjects delivered the Fit tests on time. However, the researchers in [T70] discovered that there was difficulty in learning some of the Fit fixtures, because the test subjects only used a very basic and limited number of fixtures types. The experiment performed in [T72] suggests that there was no difference in the quality of story tests produced by business graduate students or computer science graduate students. The research in [T73] suggests that experienced developers gain much more benefits from Fit tables in software evolution tasks, suggesting that previous experience does matter. The research in [T74] found that story tests alone could not communicate everything, because it didn't provide the context. The researchers in [T77] found that the story tests are the medium for communicating complex domain knowledge, especially in a very large software development team. The researchers in [T75] discovered that story tests written in Fit actually were more ambiguous to untrained test subjects, because they didn't know how to understand the Fit tables.

2.4 Code Design

The lessons learned papers identified that there are 1) a better design of the code for testability, such as separation of backend functionality from the user interface code [T15, T18, T33, T35, T38, T52, T53, T54, T55]. 2) Some discovered that the team produced quality code first time and discovered that STDD can drive quality [T21, T22, T41, T56, T71]. 3) The STDD can drive the overall code design [T17, T25,T56] and 4) developers had better understanding of their code [T45]. 5) Some papers argued that STDD also helped developers think about the user experience early [T22, T25]. There were no papers that identified the issues or concerns related to code for STDD.

There were four research papers related to the code design. The researchers in [T70] discovered that more quality code is produced the first time. The research in [T73] suggests story testing tools can help with software evolution, especially for more experienced developers who are coding alone. The researchers in [T76] confirmed that Fit tables can help developers perform code maintenance tasks correctly, because it ensures that requirements changes are implemented appropriately and the regression tests ensure that the existing functionality are not broken. However, the experiment performed by [T72] showed that there was no correlation between the quality of the story tests and the quality of the code.

2.5 Testing Tools

Many papers deal with tool support for STDD. The papers suggest that there is a lack of tools that can help facilitate STDD effectively. First, we present the discussions related to the types of tools that were used for STDD. Some used capture/replay tools [T18, T30, T51, T57, T58,]. However, most people voiced that the capture/replay tests are easily broken even with a minor/cosmetic changes in the user interface. Instead, some people use unit testing tools such as jUnit and nUnit [T15, T28, T36], because they give a lot of power to the developers for automation. Some people used word processors or spreadsheets for acquiring the story tests from the customers [T15, T18, T33, T45]. Some people used XML for the test specification [T18, T33, T43, T49, T59]. Some people preferred scripting languages or API based tools such as Selenium [T18, T28, T32, T38, T43, T55, T58]. But most people used tabular and fixture based tools such as Fit [T16, T17, T22, T25, T27, T30, T33, T34, T37, T39, T43, T44, T45, T46, T48, T49, T50, T59, T60, T61, T62].

In terms of ways people use these tools, some people argued that customers and developers ended up using different tools based on their familiarity of the tools [T21, T24, T29, T38, T45, T60]. Some people also integrated other testing, bug tracking, and/or domain-specific productivity tools [T42, T43, T60, T63].

In terms of features that people thought were important in story testing tools are automated test generation [T29, T36, T51, T58], automatic test data generation [T36, 58] and automatic documentation generation [T24, T31, T52]. In addition, some people thought important tool features include viewing the test result history [T29], refactoring of the tests [T18, T21, T25, T29, T65] and test organization features [T24, T25, T29].

In terms of research papers, [T78] analyzed whether annotated documents in story testing tools can help write better story tests. The researchers in [T79] also performed an annotation experiment on a medical domain. Their findings suggest that the participants who were given an annotation to follow created story tests with less missing elements than those groups that did not.

2.6 What to Test in STDD

We found that there are surprisingly many variations on what to test using story tests. They include the graphical user interface in order to simulate how user will interact with the system [T22, T41, T51, T57, T58], web services, web applications and network related issues [T32, T43, T49], backend functionality (functional requirements) [T31, T41], performance [T19], security [T19], stability [T19], non-functional requirements [T31, T36], end-to-end-customer's perspective of the feature [T17, T51], regression testing [T15, T21, T22, T24, T28, T32, T33, T38, T43, T48, T50, T51, T61], user interaction [T21, T25, T38, T57], concurrency [T41, T43], database [T43], only the critical features as judged by the developers [T27], and multi-layer architecture of software design [T54]. Finally, most people thought the purpose of STDD is not so much about testing, but to communicate the requirements with the customer in an unambiguous way [T17, T24, T25, T27, T29, T30, T31, T39, T55, T56, T62]. No empirical evaluation of this question exists in terms of story test driven development.

2.7 Test Automation Issues

Finally, we analyzed the issues involved with automating story tests. Some people identified that there is a real difficulty in maintaining the tests especially in large projects [T21, T24, T28, T29, T36, T51]. Similarly, there is difficulty in organizing and sorting the tests in order to see the big picture [T20, T21, T34, T38]. Some found that it is difficult to locate defective code [T18, T20, T21, T24], because a story was concerned with a bigger scope of a feature. There is a desire to automate at the user interface level [T18, T21, T25, T38, T40]. One author desires for better usability of the testing tools [T33]. Some people desired for more readable test specifications [T17, T21, T24, T25, T33, T34, T41, T42, T57, T66, T81]. Some people thought keeping track of the history of the tests is important [T34]. One author worried that the team ignored the tests because there were too many false alarms [T38], mainly because the tests relied on the GUI, which broke the tests easily even with only small changes to the user interface. Another concern is a lack of readily usable testing tools that can accommodate specific needs [T38, T45]. One author argued that the problem is with the incompatibility of different platforms and languages for the tests [T45]. Some people emphasized that tests should be written using more reusable objects and services [T34, T45, T61]. Some people argued for separation of test data and test code and the tool should help with the separation [T18, T32, T53]. No research paper analyzes these issues in relation to story test driven development.

3. Discussion

The review suggests that we need a lot more research done in the area of story test driven development. Just from the number of papers we found, we were able to discover 49 lessons learned papers and 8 tool development papers but only 8 research papers. There is no research done in the area of cost, test automation issues, or what to test. Much of the research done so far since the introduction of STDD focused on time, people and code design. The research is also done with a very small group of research participants and we still lack enough evidence to aggregate these findings into general knowledge.

We need better understanding of what can be tested by story tests and what should be outside the scope of STDD. The industry practitioners have a lot of concerns about the test automation and the lack of tools to facilitate STDD in a cost-effective and timely manner. Our findings suggest that the gap is very wide between what researchers were able to provide to the practitioners at the time and what are needed. Existing papers focused on time, people, code design and tools, but there are still many unanswered questions as well as conflicting results. Thus, a substantial amount of empirical research on STDD is needed to create a solid foundation for rational decisions in this area.

Acknowledgement: This research is supported by the National Research Council of Canada (NSERC) and iCore.

References

1. Beck, K., Extreme Programming Explained: Embrace Change, 1/e, Addison-Wesley (1999)
2. Fowler, M., Specification by Example, http://www.martinfowler.com/bliki/SpecificationByExample.html
3. Kaner, C., “Cem Kaner on Scenario Testing: The Power of ‘What-If…’ and Nine Ways to Fuel Your Imagination, Better Software, 5(5), 16-22, 2003
4. Melnik, G., Empirical Analyses of Executable Acceptance Test Driven Development, University of Calgary, PhD Thesis, 2007
5. Park, S., Maurer, F., Communicating Domain Knowledge in Executable Acceptance Test Driven Development, *XP2009, LNBIP 31,* May 2009, pp. 23-32
6. Behavior driven development, www.behaviour-driven.org
7. Park, S., Maurer, F., An Extended Literature Review on Story Test Driven Development, Technical Report, University of Calgary, 2010-953-02