An Extended Review on Story Test Driven Development

Shelly Park, Frank Maurer
Department of Computer Science
Univeresity of Calgary
{sshpark, fmaurer}@ucalgary.ca

Abstract. This paper presents a systematic review of studies 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 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, 6] and behavior driven development [7] 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 using a systematic review process. 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.

The primary audience for our research paper is researchers in the STDD field in Agile development environments and the practitioners who are pioneering the STDD process. We performed a systematic review on STDD papers (written in English) published in conferences, magazines and journals. We categorized them into lessons learned/experience papers, tool development papers and empirical research papers.

Section 2 presents the background on STDD. Section 3 presents our research design. Section 4 presents our findings. Section 5 discusses the significance of our research and section 6 presents the threats to validity in our research design. Section 7 concludes the paper with some final thoughts on our review and presents the next step in our research.

2 Story-Test Driven Development

Beck states in his book, Extreme Programming Explained [1], that software development has too much wasteful overproduction, such as requirements documents that rapidly grow obsolete, documentations that nobody reads, elaborate architecture that are never used, code that goes months without being integrated, tested and executed. He proposes that a way to improve the situation is not with more elaborate requirements-gathering processes, but rather "shortening the path between the production of requirements detail and the deployment of software specified"[1]. Beck also doesn't feel that the word, "requirements", is an appropriate terminology. Since most of these features weren't mandatory, he proposes to use the word, 'story'. A story is requirements that are broken down into smaller features that can be estimated.

In agile software engineering, the dominant form of communicating the requirements is user stories. Cohn states that "extensive upfront requirements gathering and documentation can kill a project"[8]. One of the reasons is that the requirements document becomes the goal, not the code. Therefore, user stories are meant to write down just enough so that we don't forget and we have something to estimate and plan the implementation. However, the stories alone are not enough.

Beck argues that defects destroy the trust. Therefore, he proposed Test-Driven Development (TDD). Beck proposed writing two types of tests: programmer perspective tests and customer perspective tests. Programmers can write the tests, but it will only show the programmer's perspective of the system. Therefore, another set of tests must be written from the customer's perspective. These tests can also help double check the two types of tests against each other to see if there are problems that are uncaught.

Kerievsky describes story testing in more detail [9] as "the process of providing the input data, initiating a process that corresponds to a story being tested and comparing the actual output with the expected output at the end of the process". Kerievsky also states that story tests are "most useful when automated, as this empowers customers and developers to launch them at the press of a button and discover the system's state.

Kerievsky suggests that finding the right input and expected output data requires domain knowledge, but turning them into tests require testing knowledge. Therefore, story tests may require domain experts/subject matter experts and quality assurance

experts. Unlike TDD where the only people who need to adopt are the developers, an STDD process requires people from all functional backgrounds to participate.

Marick suggests that these tests are for exploration [10] rather than testing. Therefore, he likes to call STDD "*example-driven development"* or "*business-facing tests"* instead. The purpose of the tests is to create examples that will help all stakeholders understand the domain. Getting the tests precisely right isn't the point in the beginning, because coming up with tests may require more understanding. The tests evolve with better understanding as the implementation begins. STDD is a communication tool for transferring knowledge between customers and developers.

Fowler calls this process, 'Specification by Example' [2]. Fowler suggests that specifications convey the connotation that they should be general and cover all cases. On the other hand, specification by examples mean highlighting only a few points and "you have to infer the generalizations yourself". Fowler suggests that the dominant idea with rigorous specification (and formal specifications) is that pre- and post-conditions must be explicitly stated in the requirements. However, Fowler found that pre-post conditions are very difficult to write in many situations. But asking for examples is much easier in some situations. Fowler stated that specification by examples is "less valuable in theory but more valuable in practice".

We presented some of the guiding visions of what STDD is, as they are envisioned by the industrial pioneers. However, we wanted to find out how these visions were adopted by the practitioners and what their experiences are. In addition, we want to determine what research has discovered about the STDD process.

3 Research Method

We are using a systematic review method for our analysis [11]. A systematic review is "a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area or phenomenon of interest" [11]. This research method grows out of the medical research domain, primarily because of their need to put many different empirical medical findings into a generalized understanding about the disease and treatment. The process has gained wider acceptance in other fields. In [11], Kitchenham provides an outline of what systematic review method can mean in software engineering discipline.

In a systematic review, the individual literature used for the purpose of systematic review is called *primary studies* and the systematic review is a *secondary study.* The most common reasons for using a systematic review are 1) to summarize the existing evidence, 2) to identify gaps in the current research, 3) to provide a framework in order to position new research activities and 4) to examine the extent to which empirical evidence supports or contradicts theoretical hypotheses. Our purpose for the systematic review of the STDD literature is to identify the gaps in the current research and examine the extent to which people's interpretation of the STDD process aligns with the motivations for STDD. In this section, we explore the objectives, protocols, inclusion/exclusion criteria, search strategy, quality assessment, and data extraction method and data synthesis technique.

STDD is a relatively new field of study in software engineering. To our best knowledge, no systematic review on story-test driven development has previously been published. Furthermore, most published papers so far are experience reports/lessons learned industry papers but there are not many research papers that critically examine these claims. Therefore, our objective is to find out what the industry practitioners claim to work and don't work and then compare how many of these claims were researched critically. Our review provides the areas in which more empirical research is required to either support or refute various opinions suggested by practitioners.

We collected all papers related to story-test driven development that are published in conference proceedings, magazines and journals. We only included papers written in English. We included papers published from 2001 up to September 2009. 2001 was the earliest paper that we could find. Then we categorized these publications into lessons-learned experience reports, tool development and research papers. To be categorized as a research paper, it needs to pass a quality threshold regarding the evidence included in the paper. Any papers that cannot pass the quality threshold are considered non-research papers. From these non-research papers, we divided the papers into lessons learned papers and tool development papers. We included both qualitative and quantitative studies for the research papers. We excluded papers that did not focus on agile software development or the automation of story tests.

We searched the ACM Digital Library, IEEE Xplore, ScienceDirect, SpringerLink and Google Scholar. We also manually searched the conference proceedings for XP, XP/Agile Universe and Agile conference. We also searched the web pages of the researchers and practitioners who previously published papers in story-test driven development to find any papers that were published outside of these venues. We used the following search terms: agile AND story AND test; agile AND functional AND test; agile AND acceptance AND test; agile AND customer AND test; agile AND Fit. Then we replaced the word agile with XP and Scrum for additional search. We read the entire paper and categorized them based on the quality criteria.

Quality criteria are important in order to provide inclusion/exclusion criteria, to provide a weight for the importance of the study's results, to guide the interpretations of the findings and to guide recommendations for further research. Dyba and Dinsoyr used the Critical Appraisal Skills Programme (CASP)[12][13]. The criteria are composed of 11 dimensions. They are as follows:

1) Is the paper research or a lessons learned report based on expert opinion?

2) Is there a clear statement of the aim of the research?

3) Is there an adequate description of the context in which the research was carried out?

4) Was the research design appropriate?

5) Was the recruitment strategy appropriate?

6) Was there a control group?

7) Was the data collected in ways that address the research issue?

8) Was data analysis sufficient?

9) Has the relationship between researcher and participants been considered to an adequate degree?

10) Is there a clear statement of findings?

11) Is the study of value for research or practice?

Because we considered both qualitative and quantitative review, we decided that a study need not have a control group, but it needs to specify the internal and external validity of their research.

We collected 49 lessons learned papers, 8 tool development papers and 8 research papers for our systematic review. We extracted the purpose, settings, research methods, findings of the research. We extracted the motivation for story-test driven development, proposed benefits and issues encountered from the lessons learned and tool development papers. The clarity of the defined criteria was evaluated by comparing the evaluations of a few randomly assigned papers between the two authors.

The next step is to aggregate our findings. However, an aggregation is difficult because there are no replicated studies and only very few high-quality empirical studies. Therefore, we can't perform a quantitative meta-analysis. Therefore, we used meta-ethnographic methods [14]. The main concepts were extracted from the papers in the author's terms. Then the concepts were compared to find if there are higher-order concepts (or themes) that can describe the findings into higher-order interpretations. In a meta-ethnographic synthesis, studies can be related in three ways: 1) directly comparable, 2) stand in opposition to one another, 3) they form an argument when they are taken together. The analysis provides a set of themes from the papers. From those themes, the analysis allows the investigators to find out which themes were supported by empirical evidences and which lack any empirical evidence. The comparison can show the gaps in the research evidences for further research.

4. Results

This section describes in details of the discussion points. We categorized the discussion points into 7 themes: cost, time, people, code design, testing tools, what to test, and test automation issues. After describing the points from the lessons learned and tool development papers, we also describe the findings from the research papers. Then we describe whether the research paper supports the points described in the lessons learned papers.

4.1 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. We first present the points from the lessons learned papers and tool development papers. Authors in [15, 16, 17] suggested that the benefit of STDD is to help keep the project within budget. Finsterwalder states that "the concrete feedback about the current state of the system is priceless. The team's continual small adjustments (on time) keep the project on course on time and on budget"[L3]. Schwartz also states that the automated story tests can "run often and facilitate regression testing at low cost"[16].

However, four papers [15, 16, 18, 19,] stated that STDD may not pay off because the cost of writing and maintaining the tests is high. For example, Crispin states that the QA's are "paid to be cost-effective, [but] there are cases where automating a test and running it repeatedly will not pay off in the form of defects found."[18]. In addition, four papers stated that their teams did not have the budget necessary to automate the tests [18, 19, 20, 21]. Andrea stated that "given the size and complexity of the system, this budget was not sufficient to automate acceptance tests for the entire system, so the developer and customer collaborated to define smallest possible set of representative tests for the highest priority."[20].

There were no research papers that explicitly analyzed the cost and budget aspect of STDD process or the tools.

4.2 Time

Time is important for project managers, because it has impact on the amount of resources required to complete the project. Five points were discussed in the lessons learned papers as the benefits of STDD process: 1) The STDD can help check the overall progress [15, 17, 18, 19, 22, 23, 24, 25, 26, 27, 28, 29, 68]; 2) adapt to requirements changes with the help of instant feedback, which can help keep the project on time [18, 27, 30]; 3) continuous verification (test anytime, more often, repeatedly)[17, 25, 27]; 4) better estimation of the stories [23, 31]; 5) immediate defect fixes [28, 32]. For example, Rogers [24] states that "showing the results of these tests is still important for the customer so that she can track the progress of development". Hanssen and Haugset [27] stated that the motivation for their STDD process was that "the paradigm of agile development relies on instant feedback and short development cycles; automation of acceptance tests may thus be seen as a promising initiative to ease and speed up this process". Kongsli [32] stated that STDD is "excellent for regression testing and allow for continuous integration, in turn enabling issues to be handled immediately when they appear."

However, some lessons learned papers identified three issues related to time: 1) writing and maintaining tests took considerable time [17, 28, 29, 33, 34, 35], 2) it can take long time to execute the tests [24, 34, 36, 37, 38], and 3) there can be a lack of time to build the necessary testing tools and infrastructures [28]. For example, Ghandi et al. [34] found that they had "an imbalanced team, and this forced our analysts to focus all their effort on just doing enough to keep the developers busy; and as our schedule tightened, the management team began to speculate about moving the story test writing and automation until after the story implementation." Andersson [36] discovered that "because running all tests at every build would take too long, developers pick a time up to 15 minutes and run all tests that take less than that time, before checking in". Stolberg [28] stated that he "worked on a small team and didn't seem to have any 'extra' time for [him] to work on the infrastructure [he] needed".

There were two research papers that dealt with time. The research paper, [72], discovered that the test subjects were able to write and test using story tests within an expected amount of time. They originally expected each person to contribute about 4 hours a week and most people were able to do so within the allotted time frame. It

suggests that time may not be an issue if the developers allocate appropriate time and have the guidance to complete them. The research paper, [74], discovered that timing was a matter of discipline more than an actual timing problem.

4.3 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 [17, 25, 31, 39, 40, 41, 42, 43, 63, 64], 2) confidence about the progress and deliverables [17, 19, 28, 32, 38, 39, 41, 44, 45], 3) better awareness for testing in the team [81, 35, 61, 17, 28, 68], 4) encouragement of collaboration between right people [39] and 5) anyone can quickly understand what's been developed [34, 35]. For example, Abath [41] states that "the approach presents a number of benefits, which include an effective bridging of communication gaps between clients and developers, synchronization between changes in requirements and the code written, a boost of confidence on the software that is being developed and automatically enforced focus on the client's interests, preventing feature creep." Talby et al. [35] states that "because developers were responsible for writing tests for each new feature, their test awareness increased and they prevented or quickly caught more edge cases while they worked." Ghandi et al. [34] stated that "midway through our project, the number of developers increased from 6 to 24 in approximately 4 weeks; this massive scaling was surprisingly successful – we think in part due to our use of FIT documents."

The lessons learned papers also identified two problems related to people. 1) The STDD affects everyone, which made the adoption difficult [24]. [24] states "acceptance testing is especially challenging because of the size and scope of its impact on all members of the team." 2) Some papers identified that there was no direct contact between developers and customers because the tests were too good and too explicit [34, 41]. For example [34] states that unintended side effects of STDD were that developers "will write code simply to make the tests pass without closely collaborating with the original customer to deliver the story's business value".

In addition, there were some papers that discussed about the people's skills. Some papers argued that it took too long to learn the testing tool or the specification language [27, 46, 55]. For example, [46] states "we have had, and continue to have, problems in automating acceptance tests; this is partly due to the nature of project, but also due to both unfamiliarity with the technique and lack of appropriate testing infrastructure". Some authors identified that lack of test automation experience in the team was the barrier [25, 43, 46, 48, 69], but most of them overcame the problem quickly. For example, [25] states it is difficult to "assemble a team with all the needed skills to support high-quality story test development".

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 [19, 26, 35, 48], or a separate group of dedicated developers/testers were created for STDD [18, 34]. One team used pair story testing method [18]. 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 [23, 24, 25, 27, 30, 36, 42, 49, 50]. In some cases, developers wrote the story tests with the customer collaboration [20, 26, 27, 51]. In some teams, the QAs wrote the tests in collaboration with the customer [19, 45].

We found seven research papers that looked into people related issues. [67] performed an experiment on how quickly developers can learn to use a STDD tool. [67] discovered that "FIT[80] tests describing customer requirement can be easily understood and implemented by a developer with little background on this framework". They discovered that 90% of the test subjects delivered the Fit tests. However, the researchers in [70] 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 [72] suggests that there was no difference in the quality of story tests produced by business graduate students or computer science graduate students. However, the computer science graduate students produced much more negative tests. Both business and computer science graduate students struggled with learning Fit initially and there was no correlation between prior work experience and the ability to learn Fit. However, once they learned Fit, both types of students used the tool easily and produced good quality specifications. One significant finding from [72] is that the team where the business and computer science graduate students were put into one team produced much better specifications than the teams with only computer science graduate students.

On the contrary, the research in [73] suggests that experienced developers gain much more benefits from Fit tables in software evolution tasks, suggesting that previous experience does matter. It suggests that amount of existing skills does influence the amount of benefits one can get from story testing tools.

The research in [74] found that story tests alone could not communicate everything, because it didn't provide the context. The story tests, however, encouraged more collaboration and encouraged "continuous learning about the domain and the system through testing". The researchers in [77] found that the story tests are the medium for communicating complex domain knowledge, especially in a very large software development team. It is impossible to teach the developers complex domain knowledge, but the story tests can guide the developers to implement correct functionality and seek out the necessary domain experts when the need arises.

The researchers in [75] 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. The research participant also took more time than expected to understand the requirements written in Fit. Therefore, story testing tools, such as Fit, do not necessarily guarantee improvement in communication if the users are not trained in the tool.

The research done in people-related issues on STDD at the moment provides a mixed result. However, the research tends to support the notion that the existing tools are not intuitive to use without some training. The research also seems to confirm that collaboration between subject matter experts and developers is a good practice.

4.4 Code Design

The lessons learned papers identified five benefits for code design. They stated that there is 1) a better design of the code for testability, such as separation of backend functionality from the user interface code [15, 18, 33, 35, 38, 52, 53, 54, 55]. For example, Kongsli [55] stated that "using fully automated acceptance tests entails a particular style of development that produces 'testable' code." 2) Some discovered that the team produced quality code the first time and discovered that STDD can drive quality [21, 22, 41, 56, 71]. For example, [41] found that "fewer bugs were discovered when the system was placed in production." 3) The STDD can drive the overall code design [17, 25, 56] and 4) developers had a better understanding of their code [45]. For example, Abbattista [45] found that the team had "better understanding of the system to be migrated and a valid starting to point make a migration plan" because of STDD. 5) Some papers argued that STDD also helped developers think about the user experience early [22, 25]. There were no papers that identified issues or concerns related to code for STDD.

There were four research papers related to the code design. The researchers in [70] discovered that more quality code is produced the first time. The research in [73] suggests story testing tools can help with software evolution, especially for more experienced developers who are coding alone. However, the benefit of Fit tables in software evolution tasks decrease when the developers are working in pairs. The researchers in [76] 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 functionalities are not broken. However, the experiment performed by [72] showed that there was no correlation between the quality of the story tests and the quality of the code. It suggests that story tests are not a good tool for controlling the quality of code.

4.5 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 [18, 30, 51, 57, 58,]. However, there is clear disadvantage with these tools because the GUI must exist in order to create the tests. 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 [15, 28, 36], 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 [15, 18, 33, 45]. Some people used XML for the test specification [18, 33, 43, 49, 59]. Some people preferred scripting languages or API based tools such as Selenium [18, 28, 32, 38, 43, 55, 58]. But most people used tabular and fixture based tools such as Fit [16, 17, 22, 25, 27, 30, 33, 34, 37, 39, 43, 44, 45, 46, 48, 49, 50, 59, 60, 61, 62].

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 [21, 24, 29, 38, 45, 60]. For example, [24] states "customers, however, don't use an IDE; now while you can teach them to use an IDE, which is something that I have tried on a previous project, it is advisable to enable customers to use a tool that they are familiar with or that is easily accessible to them." Some people also integrated other testing, bug tracking, and/or domain-specific productivity tools [42, 43, 60, 63]. For example, [60] integrated MatLab to work with Fit and [63] integrated wiki and Mantis to their STDD tool. Some people felt there was a need to integrate with distributed automation framework such as STAF [28, 43]. In summary, it seems that there is a need for STDD tools to be integrated with many different types of tools so that users can define tests in their familiar tools.

In terms of features that people thought were important in story testing tools are automated test generation [29, 36, 51, 58], automatic test data generation [36, 58] and automatic documentation generation [24, 31, 52]. Automatic test generation could mean automatically creating test fixtures from Fit tables [29] or creating tests from sample web pages by tracing user inputs from a web page [51]. For automatic test data generation, [36] tried to mine input data provided by the customers and feeding them into the tests automatically. Some people argued that story tests could automatically turned into user manuals. [52] developed a tool that can automatically output an "English translation" of the tests into an HTML file.

In addition, some people thought important tool features include viewing the test result history [29], refactoring of the tests [18, 21, 25, 29, 65] and test organization features [24, 25, 29]. For example, [65] provides an ability to automatically refactor story tests. [25] desires that tools could "manage and organize very large suites of story tests to make it easier to find those that are relevant to a particular persona, user task, interaction context, use cases, or domain object for example; keeping the story tests consistent with the underlying assumptions".

In terms of research papers, [78] analyzed whether annotated documents in story testing tools can help write better story tests. The annotations are pre-defined keywords that must be used for the testing tool for parsing out the tests properly. The test subjects had a central tendency to agree mostly. The researchers in [79] 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. The research supports the use of annotations in the story test tools.

4.6 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 [22, 41, 51, 57, 58], web services, web applications and network related issues [32, 43, 49], backend functionality (functional requirements) [31, 41], performance [19], security [19], stability [19], non-functional requirements [31, 36], end-to-end-customer's perspective of the feature [17, 51], regression testing

[15, 21, 22, 24, 28, 32, 33, 38, 43, 48, 50, 51, 61], user interaction [21, 25, 38, 57], concurrency [41, 43], database [43], only the critical features as judged by the developers [27], and multi-layer architecture of software design [54]. Finally, most people thought the purpose is not so much about testing, but to communicate the requirements with the customer in an unambiguous way [17, 24, 25, 27, 29, 30, 31, 39, 55, 56, 62].

No empirical evaluation of this question exists.

4.7 Test Automation Issues

Finally, we analyzed the issues involved with automating story tests. Some people identified that there is difficulty in maintaining the tests especially in large projects [21, 24, 28, 29, 36, 51]. For example, [21] states that currently story tests "don't have the same type of regression safety net as production code", which makes it difficult to safely make changes to the story tests without introducing unintended interactions between the tests. Similarly, there is difficulty in organizing and sorting the tests in order to see the big picture [20, 21, 34, 38]. For example, [38] states "we found that keeping all of the tests in one Suite was the easiest way to manage the tests, but that frequently we wanted to group the tests in other ways (by story, by iteration, by functionality set)" but they discovered that moving around a large set of stories was difficult. Similarly, [34] discovered that "moving the documents around different directories had its drawbacks; firstly, it was prone to error, with people forgetting to commit both the removal and addition of a moved document; this led to a confusion and time spent sorting out which was correct".

Some found that it is difficult to locate defective code [18, 20, 21, 24], because a story was concerned with a bigger scope of a feature. There is a desire to automate at the user interface level, but the authors of the papers couldn't because these tests break down easily [18, 21, 25, 38, 40]. One author desires for better usability of the testing tools [33]. Some people desired for more readable test specifications [17, 21, 24, 25, 33, 34, 41, 42, 57, 66, 81]. Some people thought keeping track of the history of the tests is important [34]. One author worried that the team ignored the tests because there were too many false alarms [38], 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 [38, 45]. One author argued that the problem is with the incompatibility of different platforms and languages for the tests [45]. Some people emphasized tests should be written using more reusable objects and services [34, 45, 61]. Some people argued for separation of test data and test code and that the tool should help with the separation of test and data better [18, 32, 53].

No research paper analyses these issues.

5. 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 in the last decade 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.

There is currently no research done looking into the cost effectiveness of STDD. There are a lot of good motivations, but there is still no concrete evidence that can help remove fears that story tests can add extra cost. In terms of time, some practitioners found that it can improve the ability to keep the project on time and help with the story estimation. Two research papers suggest that the people's fear of time is unfounded as long as appropriate amount of time is allocated for STDD.

Significant portions of research are currently investigating people related issues. However, there are some conflicting findings. The research looks specifically into Fit [80] tools and the findings suggest that Fit is easy to use only after some training. Some findings suggest that previous experience doesn't matter in producing quality story tests, but some findings suggest that previous experience does matter in how much benefit one can get out of these tests. More research is needed. Surprisingly, significant portions of the research papers focused on code design, despite the fact that story tests are not for the developers. Perhaps, researchers should focus more on the original intent of the story tests instead of focusing on the developers too much.

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 this point in time and what is needed.

Overall, we discovered that only a very limited number of research papers exist in the current literature and that the empirical basis on STDD is weak. 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. Sauve, J., Neeto, O., Teaching Software Development wit ATDD and EasyAccept, SIGCSE 2008, pp. 542-546
6. Ricca, F., Penta, M., Torchiano, M., Tonella, P., Ceccato, M., Visaggio, C., Are Fit Tables Really Talking? A Series of Experiments to Understand whether Fit Tables are Useful during Evolution Tasks, *Int. Conf. on Soft. Eng. 2008*, pp. 361-370
7. Behavior driven development, www.behaviour-driven.org
8. Cohn, M., *User Stories Applied: For Agile Software Development*, Addison-Wesley Professional, 2004
9. Kerievsky, J. Storytesting, http://industrialxp.org/storytesting.html
10. Marick, B., Example-Driven Development, http://www.exampler.com
11. Kitchenham, B., Procedure for Performing Systematic Reviews, Keele Univeresity, Technical Report TR/SE-0401, NICTA Technical Report 0400011T.1, July 2004, Australia
12. Greenhalgh, T., How to Read a Paper, 2nd ed. BMJ Publishing Group, London, 2001
13. Dybå, T., Dingsøyr, T., Empirical studies of agile software development: A systematic review, *Information and Software Technology*, 50 (2008), pp. 833-859
14. Noblit, G., Hare, R., Meta-Ethnography: Synthesizing Qualitative Studies, Sage Publications, London, 1988
15. Finsterwalder, M., Automatic Acceptance Tests for GUI Applications in an Extreme Programming Environment, *Proc. of the 2nd Int. Conf. on Extreme Prog.*, 2001, pp. 114-117
16. Schwarz, C., Skytteren, S., Ovstetun, T., Aut-AT: An Eclipse Plugin for Automatic Acceptance Testing of Web Applications, *OOPSLA* 2005, pp. 182-183
17. Haugset, B., Hanssen, G., Automated Acceptance Testing: A Literature Review and an Industrial Case Study, Proc. of Agile 2008, pp. 27-38
18. Crispin, L., House, T., Testing in the Fast Lane: Automating Acceptance Testing in an Extreme Programming Environment, XP/Universe Conference, 2001
19. Crispin, L., House, T., Wade, C., The Need for Speed: Automating Acceptance Testing in an eXtreme Prog. Env., *eXtreme Prog. and Flexible Proc. in Soft. Eng.*, 2001, pp. 96-104
20. Andrea, J., Putting a Motor on the Canoo WebTest Acceptance Testing Framework, *Proc. of the 5th International Conference on Extreme Programming*, pp. 20-28
21. Andrea, J., Envisioning the Next Generation of Functional Testing Tools, IEEE Software, May/June 2007, pp. 58-66
22. Steinberg, D., Using Instructor Written Acceptance Tests Using the Fit Framework, *LNCS*, Vol. 2675, Springer-Verlag, pp. 378-385, 2003
23. Watt, R., Leigh-Fellows, D., Acceptance Test Driven Planning, *LNCS, Vol. 3134*, Springer-Verlag, pp. 43-49, 2004
24. Rogers, R., Acceptance Testing vs. Unit Testing: A Developer's Perspective, *Proc. of Extreme Programming in Agile Methods, 2004, LNCS 3134*, pp. 22-31, 2004
25. Mulgridge, R., Managing Agile Project Requirements with Story Test Driven Development, *IEEE Software*, 25(1), pp. 68-75
26. Talby, D., Dubinsky, Y., Government of an Agile Software Project, Proc. of the 2009 ICSE Workshop on Software Development Governance, 2009, pp. 40-45
27. Hanssen, G., Haugset, B., Automated Acceptance Testing Using Fit, 42nd Hawaii International Conference on System Sciences, 2009, pp. 1-8
28. Stolberg, S., Enabling Agile Testing through Continuous Integration, Agile 2009
29. Khandkar, S., Park, S., Ghanam, Y., Maurer, F., FitClipse: A Tool for Executable Acceptance Test Driven Development, Proc. of *XP 2009*, pp. 259-260
30. Martin, R., Melnik, G., Test and Requirements, Requirements and Tests: A Mobius Strip, IEEE Software, 25(1), pp. 54-59
31. Onions, P., Patel, C., Enterprise SoBA: Large-scale Implementation of Acceptance Test Driven Story Cards, *Proc. of IEEE Int. Conf. on Inf. Reuse & Int.*, pp. 105-109, 2009
32. Kongsli, V., Towards Agile Security in Web Applications, *Proc. of the Companion to the 21st ACM SIGPLAN Symposium OOPSLA*, October 2006

33. Andrea, J., Generative Acceptance Testing for Difficult-to-Test Software, *Proc. XP 2004*, LNCS Vol. 3092, pp. 29-37, 2004
34. Gandhi, P., Haugen, N., Hill, M., Watt, R., Creating a Living Specification using FIT documents, Proc. of the Agile Conference 2005, July 24-29, pp. 253-258
35. Talby, D., Keren, A., Hazzan, O., Dubinsky, Y., Agile Software Testing in a Large-Scale Project, *IEEE Software, July/August* 2006, pp. 30-37
36. Andersson, J., Bache, G., Sutton, P., XP with Acceptance-Test Driven Development: A Rewrite Project for a Resource Optimization System, *Proc. of the 4th International Conference on Extreme Programming*, 2003, pp. 189-197
37. Mugridge, R., Cunningham, W., Agile Test Composition, *Proc. of the 6th International Conference on Extreme Programming*, 2005, LNCS 3556, pp. 137-144
38. Holmes, A., Kellogg, M., Automating Functional Tests using Selenium, *Agile Conference 2006*
39. Reppert, T., Don't Just Break Software, Make Software: How Story-Test Driven Development is Changing the Way QA, Customers, and Developers Work, *Better Software 6(6)*, 18-23, 2004
40. Stevenson, C., Pols, A., An Agile Approach to a Legacy System, *Proc. of the 5th International Conference on Extreme Programming*, 2004, LNCS 3092, pp. 123-129
41. Abath, O., Rocha, E., Sauve, J., Experience Report: Using Easy Accept to Drive Development of Software for an Energy Company, *Proc. Workshop SAST 2007*, pp. 79-84
42. Gobbo, F., Bozzolo, P. Girardi, J., Pepe, M., Learning Agile Methods in Practice: Adv. Educ. Aspects of the Varese XP-UG Experience, *XP* 2007, LNCS 4536, pp.173-174
43. Kim, E., Na, J., Ryoo, S., Developing a Test Automation Framework for Agile Development and Testing, *XP* 2009, LNBIP 31, pp.8-12
44. Muller, M., Link, J., Sand, R., Malpohl, G., Extreme Programming in Curriculum: Experiences from Academia and Industry, *XP* 2004, LNCS 3092, pp.294-302
45. Abbattista, F., Bianchi, A., Lanubile, F., A Story-Test Driven Approach to the Migration of Legacy Systems, *XP* 2009, LNBIP 31, pp. 149-154
46. Grossman, F., Bergin, H., Leip, D., Merritt, S., Gotel, O., One XP Experience: Intro. Agile (XP) Soft. Development into a Culture that is Willing But Not Ready, CASCON '04
47. Martin, D., Rooksby, J., Rouncefield, M., Sommerville, I., 'Good' Org. Reasons for 'Bad' Soft. Testing: An Ethno. Study of Test. in a Small Soft. Comp., *ICSE* 2007, 862-863
48. Sumrell, M., From Waterfall to Agile – How does a QA Team Transition?, Agile 2007
49. Mugridge, R., Tempero, E., Retrofitting an Acceptance Test Framework for Clarity, *Proc. Agile Development Conference*, 2003, pp. 92-98
50. Deng, C., Wilson, P., Maurer, F., Fitclipse: A Fit-Based Eclipse Plug-in for Executable Acceptance Test Driven Development, *XP 2007, LNCS 4536*, 93-1000
51. Mugridge, R., MacDonald, B., Roop, P., A Customer Test Generator for Web-based Systems, *XP 2003, LNCS 2675*, pp. 189-197
52. Pawson, R., Wade, V., Agile Development Using Naked Objects, *Proc. of the 4th XP 2003, LNCS 2675*, 97-103
53. Martin, R., The Test Bus Imperative: Architectures that Support Automated Acceptance Testing, *IEEE Software, July/August 2005*, pp. 65-67
54. Park, S., Maurer, F., Multi-modal Functional Test Execution, *XP 2008, LNBIP 9*, pp. 218-219, 2008
55. Kongsli, V., Security Testing with Selenium, *Proc. of Companion to the 22nd ACM SIGPLAN Conference on OOPSLA 2007*, pp. 862-863
56. Yague, A., Rodriguez, P. Garbajosa, J., Optimizing Agile Processes by Early Identification of Hidden Requirements, *XP 2009, LNBIP 31*, pp. 180-185
57. Andersson, J., Bache, G., The Video Store Revisited Yet Again: Adventure in GUI Acceptance Testing, *XP 2004, LNCS 3092*, pp. 1-10

58. Andersson, J., Bache, G., Verdoes, C., Multithreading and Web Applications: Further Adventures in Acceptance Testing, *XP 2005, LNCS 3556*, pp. 210-213
59. Nielsen, J., McMunn, D., The Agile Journey Adopting XP in a Large Financial Services Organization, *XP 2005, LNCS 3556*, pp. 28-37
60. Chen, J., Smith, M., Geras, A., Miller, J., Making Fit/FitNesse Appropriate for Biomedical Engineering Research, *XP 2006, LNCS 4044*, pp. 186-190
61. Miller, J., Smith, M., A TDD Approach to Introducing Students to Embedded Programming, ACM SIGCSE Bulletin, Vol. 39, Iss. 3, September 2007, pp. 33-37
62. Park, S., Maurer, F., The Requirements Abstraction in User Stories and Executable Acceptance Tests, Agile 2008, Toronto, Canada
63. Chubov, I., Droujkov, D., User Stories and Acceptance Tests as Negotiation Tools in Offshore Software Development, *XP 2007, LNCS 4536*, pp. 167-168
64. Park, S., Maurer, F., The Benefits and Challenges of Executable Acceptance Testing, Workshop on Scrutinizing Agile, In Conjunction with ICSE 2008
65. Ordelt, H., Maurer, F., Acceptance Test Refactoring, *XP2008*, Limerick, Ireland, Springer, 10-14 June 2008
66. Geras, A., Miller, J., Smith, M., Love, J., A Survey of Test Notations and Tools for Customer Testing, *XP2005*, LNCS 3556, pp. 109-117
67. Melnik, G., Read, K., Maurer, F., Suitability of FIT User Acceptance Tests for Specifying Func. Req.: Dev. Perspective, *XP/Agile Universe 2004*, LNCS 3134, 60-72
68. Melnik, G., Maurer, F., The Practice of Specifying Requirements Using Executable Acceptance Tests in Computer Science Courses. *OOPSLA 2005*
69. Read, K., Melnik, G., Maurer, F., Student Experiences with Executable Acceptance Testing, *Proc. of Agile 2005*, IEEE Press, 2005
70. Read, K., Melnik, G., Maurer, F., Examining Usage Patterns of the FIT Acceptance Testing Framework, *XP 2005*, LNCS 3556, 127-136
71. Sauve, J., Neto, O., Cirne W., EasyAccept: A Tool to Easily Create, Run and Drive Dev. w/Auto. Acceptance Tests, *2006 Int. Work. on Auto. of Soft. Test*, 2006, 111-117
72. Melnik, G., Maurer, F., Chiasson, M., Executable Acceptance Tests for Communicating Business Requirements: Customer Requirements, Agile 2006, 35-46
73. Ricca, F., Penta, M., Torchiano, M., Tonella, P., Ceccato, M., Visaggio, C., Are Fit Tables Really Talking?, ICSE 2008, 361-370
74. Melnik, G., Maurer, F., Multiple perspectives on Executable Acceptance Test-Driven Development, *XP 2007*, LNCS 4526, pp. 245-249, 2007
75. Ricca, F., Torchiano, M., Ceccato, M., Tonella, P., Talking tests: An Empirical Assessment of the role of fit acceptance test in clarifying requirements, IWPSE 2007, 51-58
76. Ricca, F., Torchiano, M., Di Penta, M., Ceccato, M., Tonella, P., The user of executable fit tables to support maint. and evol. tasks, Elect. Comm. of the EASST, 8, 2008
77. Park, S., Maurer, F., Communicating Domain Knowledge in Executable Acceptance Test Driven Development, *XP2009, LNBIP 31*, May 2009, pp. 23-32
78. Connolly, D., Keenan, F., McCaffery, F., Developing acceptance tests from existing docum. using annot.: An Experiment, *ICSE Works. on Auto. of Soft. Test*, 2009, 123 – 129
79. Connolly, D., McCaffery, F., Keenan, F., Automating Expert-Defined Tests: A Suitable App. for the Medical Device Ind.?, *Euro. Conf. on Soft. Proc. Imp.*, 2009, 32-43
80. Mugridge, R., Cunningham, W., *Fit for Developing Software: Framework for Integrated Tests*, Prentice Hall, 2005
81. Reichlmayr, T., The agile approach in an undergraduate software engineering course project, *33rd Annual Frontiers in Education, 2003*, pp. 13-18, S2C Vol. 3