UNIVERSITY OF CALGARY

Communicating Domain Knowledge through Example-Driven Story Testing

by

Shelly S. Park

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

September, 2011

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Communicating Domain Knowledge through Example-Driven Story Testing" submitted by Shelly Park in partial fulfilment of the requirements of the degree of Doctor of Philosophy.

Supervisor, Dr. Frank Maurer, Department of Computer Science, University of Calgary

---

Dr. Armin Eberlein, Department of Computer Science & Engineering, American University of Sharjah

---

Dr. Daniela Damian, Department of Computer Science, University of Victoria

---

Dr. Xin Wang, Department of Geomatics Engineering, University of Calgary

---

DOCTORAL STUDENTS ONLY External Examiner (or External Reader), Dr. Robert Biddle, School of Computer Science, Carleton University

---

Date

Abstract

This dissertation investigates the uses of Story Test Driven Development in Agile software development teams. There are three main research questions: 1) What problems are faced by Agile teams in practicing Story Test Driven Development? 2) Investigate the relationship between stories, teams and defects. 3) What are the factors that lead to successful adoption of Story Test Driven Development? In this dissertation, we explore these questions using four case studies.

The main contribution of this research is to approach Story Test Driven Development as a knowledge building process rather than as a software testing process. The studies suggest that Story Test Driven Development is particularly useful for communicating domain knowledge between customers (domain experts) and the developers. The automated testing aspect of the story tests allows developers to implicitly learn and directly validate their understanding of the domain knowledge and the requirements. Story tests are not a software testing tool, but a validation tool about how domain knowledge and other requirements should be implemented in software.

In addition, we discovered that the main bottleneck in the successful adoption of Story Test Driven Development is the customer participation. Story Test Driven Development is a way for customers to engage in software product creation in a much more direct way. There should be a community of contributors and personal rewards for contributing the story tests. The contributors exhibit “selfish altruism” in their motivation for participation. The success of Story Test Driven Development is not in producing better software testing methods but in fostering the community of contributors.

Acknowledgements

- I would like to thank my supervisors, Dr. Frank Maurer, Dr. Armin Eberlein and Dr. Daniela Damian for their invaluable feedback, their support and advices.
- This research is supported by following scholarships and grants. I would like to thank them for their financial support.
  - NSERC Postgraduate Scholarships
  - iCore PhD Scholarship
  - Departmental Research Awards
  - University of Calgary Graduate Travel Grants
  - Queen Elizabeth II Doctoral Scholarship
  - Agile Academic Grant
  - Alberta Graduate Scholarship
- Finally, I would like to thank my parents for their encouragement.

Related Publications and Presentations

- Park, S., Maurer, F., Eberlein, A., Fung, T-s., (2010) Requirements Attributes to Predict Requirements Related Defects, 20th IBM Annual International Conference Centre for Advanced Studies Research, Toronto, Canada, Nov.7-10, 2010
- Park, S., Maurer, F (2010) A Network Analysis of Stakeholders in Tool Visioning Process for Story Test Driven Development, IEEE ICECCS 2010 15th International Conference on Engineering of Complex Computer Systems, St. Anne's College, Oxford, United Kingdom, March 22-26, 2010
- Shelly Park and Frank Maurer, A Literature Survey on Story Test Driven Development. Proc. of 11th International Conference on Agile Processes and eXtreme Programming (XP 2010), Trondheim, Norway, 2010
- Park, S., Maurer, F (2009) "Communicating Requirement Domain Knowledge in Executable Acceptance Test Driven Development", In Proc. of 10th International Conference on Agile Processes and eXtreme Programming (XP 2009), Pula, Sardinia, Italy, pp. 23-32
- Park, S., Maurer, F. (2009) "The Role of Blogging in Generating a Software Product Vision", In CHASE 2009 workshop, Collocated with 31st International Conference on Software Engineering (ICSE 2009), Vancouver, Canada
- Park, S., Maurer, F., (2009) "AP Jazz: Integrating Synchronous Distributed Project Planning with Executable Acceptance Test Driven Development for Agile Software Teams", IBM Jazz Event at 31st International Conference on Software Engineering (ICSE 2009), Vancouver, Canada, May 19, 2009

- Park, S., Maurer, F (2009) “Communicating Requirement Domain Knowledge in Executable Acceptance Test Driven Development”, In Proc. of 10th International Conference on Agile Processes and eXtreme Programming (XP 2009), Pula, Sardinia, Italy
- Khandkar, S., Park, S., Ghanam, Y., Maurer, F. (2009) “FitClipse: A Tool for Executable Acceptance Test Driven Development”, In Proc. of 10th International Conference on Agile Processes and eXtreme Programming (XP 2009), Pula, Sardinia, Italy, pp. 259-260
- Wang, X., Ghanam, Y., Park, S., Maurer, F. (2009) “Using Digital Tabletops to Support Distributed Agile Planning Meetings”, In Proc. of 10th International Conference on Agile Processes and eXtreme Programming (XP 2009), Pula, Sardinia, Italy
- Hosseini-Khayat, A., Ghanam, Y., Park, S., Maurer, F. (2009) “ActiveStory Enhanced: Low-Fidelity Prototyping and Wizard of Oz Usability Testing Tool”, In Proc. of 10th International Conference on Agile Processes and eXtreme Programming (XP 2009), Pula, Sardinia, Italy
- Park, S., Maurer, F. (2009) A Network Analysis of Online Forum Discussions on Executable Acceptance Test Driven Development, University of Calgary, Department of Computer Science, 2009-929-08, May 12, 2009
- Park, S., Maurer, F. (2009) "A Network Analysis of Online Forum Discussions on Executable Acceptance Test Driven Development", Technical Report, 2009-929-08, University of Calgary, Department of Computer Science

- Park, S., Maurer, F. (2008) “The Requirements Abstraction in User Stories and Executable Acceptance Tests”, Agile 2008, Toronto, Canada
- Park, S., Maurer, F. (2008) “The Application of Multi-modal Test Execution Using Fitclipse”, Agile 2008, Toronto, Canada
- Park, S., Maurer,F. (2008) “Multi-modal Functional Test Execution”, In Proc. of 9th International Conference on Agile Processes and eXtreme Programming (XP 2008), Limerick, Ireland, pp. 218-219 (Acceptance Rate: 24%)
- Park, S., Maurer, F. (2008) “Benefits and Challenges of Executable Acceptance Testing”, In Proc. of APSO 2008 Workshop, Collocated with 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, pp. 19-22
- Ghanam, Y., Park, S., Maurer, F. (2008) “A Test-Driven Approach to Establishing & Managing Agile Product Lines”, In Proc. of the 5th Software Product Lines Testing Workshop (SPLiT 2008) in conjunction with SPLC 2008, Limerick, Ireland, pp 151-156
- Nehring, K., Park, S., Maurer, F. (2008) “Leveraging the Jazz Platform for Developing an Agile Planning Tool”, Infrastructure for Research in Collaborative Software Engineering 2008 (iReCoSe 2008), in Conjunction with FSE 2008, Atlanta, Georgia, USA
- Park, S. (2008) “Multi-modal Acceptance Testing”, Agile Alliance Functional Testing Tools Workshop, Toronto, Canada, Aug 2008
- Park, S., Maurer, F. (2008) “The Requirements Abstraction in User Stories and Executable Acceptance Tests”, Ideaca Calgary, Jul 18, 2008

- Park, S., Maurer, F. (2008) “The Application of Multi-modal Test Execution Using Fitclipse”, Ideaca Calgary, Jul 18, 2008

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
API	Application Programming Interface
AT	Acceptance Tests
CASP	Critical Appraisal Skills Programme
DOI	Diffusion of Innovation
EATDD	Executable Acceptance Test Driven Development
GUI	Graphical User Interface
IDE	Integrated Development Environment
STDD	Story Test Driven Development
QA	Quality Assurance Analyst/Engineer
ROI	Return on Investment
RUP	Rational Unified Process
UI	User Interface
XP	Extreme Programming

CHAPTER 1: INTRODUCTION

1.1 Introduction

This dissertation investigates the uses of Story Test Driven Development in Agile software development teams. Story Test Driven Development (STDD) or Executable Acceptance Driven Development (EATDD) is a way of communicating requirements through automated tests. It belongs to the Agile software engineering methodology and its purpose is to communicate requirements more effectively using specifications that can be automatically tested against the code. Currently, Story Test Driven Development is still in its formative stage and many ideas are being put forward by the community. This concept is called by many names - customer tests [B99], functional tests [K11], story tests [K11], executable acceptance tests [K11], example-driven development [Ma11], scenario tests [K03] and specifications by example [Fo11] among many more. This dissertation will refer them as *story tests* and *Story Test Driven Development* where story tests are the artefacts for communicating requirements and Story Test Driven Development is the process of using these artefacts to facilitate the software development.

1.2 Definitions

Before we begin, we need to introduce the following key terms as they will be used often in the dissertation. They have specific meanings in Agile software engineering. Therefore, we need to make sure that these terms are applied within the context of Agile software engineering.

- **Acceptance Test:** Another name for Story Test. These tests are written by customers to convey the software requirements in a form of tests that can either pass or fail. The developers create the automated tests to show that the code passes these tests according to the way customers specified them.
- **Business Experts (Domain Experts):** The person who has the customer's domain knowledge. They are often just referred to as customers.
- **Customer:** People who are responsible for representing the end-users and/or sponsors who are paying for software development. Or they are the actual end-users and/or sponsors who are paying for the software development. They may also refer to any stakeholders who are not part of the development team. They are responsible for writing the story tests.
- **Developer:** They are responsible for producing software. They are responsible for writing the code that automates the story tests and passing the story tests.
- **Executable Acceptance Test:** Another term for the automated story tests, These are acceptance tests with the executable test code. Often people just called them acceptance test for short.
- **Executable Acceptance Test Driven Development:** Another term for Story Test Driven Development. It is the process of using story tests facilitates the software development. The customers write the story tests and the developers write the code that can pass the story tests.
- **Story (User Story):** The user story is a feature that can be implemented in software written from the customer's point of view. It can contain

functional and non-functional requirements. Each story is estimated by the developer who will implement the feature. See the detailed discussion on its definition in Chapter 2.4

- **Story Test:** These tests are written by customers to convey the software requirements in a form of tests that can either pass or fail. The developers create the automated tests to show that the code passes these story tests according to the way customers specified them.
- **Story Test Driven Development:** It is the process of using story tests facilitates the software development. The customers write the story tests and the developers write the code that can pass the story tests.
- **Test Driven Development:** A software development process where the developers write the tests first then write the code that passes the tests. The process usually refers to the unit tests.

1.3 A Brief Introduction to Agile Methodologies

Agile methodologies have grown and matured very quickly within the last decade. Story Test Driven Development is a new requirements engineering technique within Agile software engineering. We collected the community's vision for the Story Test Driven Development in Chapter 4, but it still lacks the foundations to solidify what Story Test Driven Development is and how one can practice it. Story Test Driven Development is not about competing with traditional requirements engineering methodology. Rather, the focus is on accommodating requirements engineering methods for teams that practice Agile methodologies and its principles. Despite some confusion, Agile methodologies

have different management theories behind their principles and practices than traditional methodologies. Therefore, it is inherently difficult to compare and contrast requirements engineering methodologies that are meant to be applied in different team organizations, different development practices, and different principles.

Requirements engineering has been one of the most misunderstood parts of Agile methodologies, mainly because the Agile principles state that their values working code over documentations. Some people interpreted it to mean that there is no requirement engineering in Agile methodologies, because there is no documentation, which is simply not true. Requirements engineering is less formulated and still a less researched part of the Agile methodologies, but it has a set of theories that guide how requirements should be solicited. The fundamental difference between traditional software engineering and Agile software engineering is that Agile methods do not approach requirements engineering as a separate task and a separate phase of the development. Rather, requirements engineering is embedded in the overall practice. Therefore, in order to explain how requirements engineering works in Agile methods, we need to explain the whole methodology. In Chapter 2, we will provide a more in-depth review of how Agile methodologies came about and how requirements engineering is viewed within the Agile context. However, for now, we introduce the topic briefly.

The methodologies that fall into Agile methodology category uphold the Agile principles as they are stated in Agile Manifesto [A11]:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation

- Responding to change over following a plan

The Agile methodologies are designed to check and balance through their iterative and interactive approaches with these principles in mind. The problem is that people try to apply these principles selectively on traditional methodologies, while the traditional methods are not designed with these principles in mind.

There is also more than one way of doing things in Agile methodologies, as evidenced by many methodologies that comprise the area. Compared to other software engineering methodologies, Agile methodologies are much newer. For example, the Agile manifesto has not been formed and published until 2001 [A11]. However, during the last decade, it proliferated in the industry quickly and a lot of practitioners embraced the Agile methodologies. It is important to view Agile methodologies as a group of methodologies in evolution - changing as they are constantly being applied on many different teams and projects.

Agile methodologies have some techniques that were instant hits with industry practitioners. For example, test-driven development, Scrum and iterative development, just to name a few. The surprising part of their success is the simplicity of these techniques, but the benefits they provide were far greater. One of these simple but profound techniques is the use of stories. Stories are the main artefacts produced for communicating requirements. Stories contain just enough information to start the conversations on what needs to be developed. It can be written by anyone. At the start of the iteration, some of these stories are chosen, discussed and estimated before they go into development.

1.3 Research in Story Test Driven Development

Story tests are a set of tests that customers provide to the developers. The developers can make sure that they understood their customers' requirements correctly by testing their code against these story tests. This will ensure that software is delivered according to the way customer envisioned it. However, story testing has been met with a lot of confusion and met with limited adoption in the industry so far. Moreover, not many empirical studies are done on Story Test Driven Development and it still requires more scientific understanding of its process. Our research is to find out and offer a solution as to the uses of Story Test Driven Development in Agile development and how one may need to practice Story Test Driven Development.

The Agile methodology is based on the assumption that requirements will constantly change and that developers must be prepared for the changes at any point in time of the software development. Therefore, being able to quickly figure out how the requirement is changed and how this reflects on the code is important. Current research interest in Story Test Driven Development, unlike other Agile methodologies, evolves heavily around tools for writing and maintaining story tests. If the story tests fail, it means the requirements are misunderstood by the developers or another part of the requirements is broken due to the new changes in the code. By writing the requirements in a testable way that can either succeed or fail, all of the stakeholders can get the state of the software development progress automatically at any time by running these automated story tests. It is an attempt to apply test-driven development to the requirements specifications. Much like how xUnit tests [J11, N11, F11] are a vital part of the Test-Driven Development [B02], a tool is required to automate these executable

specifications. Therefore, much of the recent focus has been about developing tools for story testing [AA11].

The tools for Story Test Driven Development are influenced by software testing tools and testing approaches. Riding on the success of the test-driven development [B02], many Agile software engineers saw an opportunity to integrate all stakeholders to participate in the test-driven development process. The idea behind Story Test Driven Development is to write requirements in a testable way to minimize miscommunication between customers and developers. The automation of these specifications into tests would ensure that the implementation is verified continuously and let the customers know about the implementation progress. There are currently several tools to facilitate Story Test Driven Development, but the most popular are Fit [Fit11] and Fitnesse [Fitn11].

The Agile community has identified recently that existing tools do not support Story Test Driven Development very well [AA11]. One of the main problems we see is that these tools do not have a clear focus on what problem they are trying to solve. People wished for better tools, but the community discovered that finding the right requirements for the new tool is actually very difficult. There are many conflicting ideas and wish lists for the tool [AA11]. To find the requirements, one must analyze what kind of problems Story Test Driven Development is trying to solve in agile software development and then discover how a tool can help solve the problem.

It is fundamentally different to get testers to write the test specifications and to get requirements engineers to write the requirements in a testable form. From the testers' perspective, even if they are thinking about testing from the requirements engineering stage, their purpose of writing the tests is to find and prevent the defects at the end. From

the requirements engineers' perspective, the purpose is about communicating the requirements to the developers. Writing requirements in a testable way is not about finding the location of the code defects, but communicating whether the requirements are correctly translated into functionalities. The other obvious difference is that the people who occupy these roles have different backgrounds and trainings. People who occupy the testing roles have different set of skills than those who occupy the requirements analysis roles or customer roles. Therefore, it is important to look at Story Test Driven Development from a holistic view.

1.4 Problem Statement

A fundamental research question is what is a story test and what can we do with it? The dissertation is an exploration to discover different interpretations of story tests and how people practice Story Test Driven Development. In order to do so, we need better understanding of the people for whom these story tests are meant for and how story tests benefit their software development process. In Agile methodologies, the teams are categorized broadly into the developers and customers. The developers are the ones who write, maintain and test the code; customers are rest of the people. Story tests are supposed to be written and maintained by the people who fall into the customer category. We need to figure out the main problems and challenges that these customer groups are facing before we suggest how story tests should be written.

1.5 Research Questions

The research goal is investigate why people use Story Test Driven Development in Agile software development. There are three main research questions:

Research Question 1: 1) What problems are faced by Agile teams in practicing Story Test Driven Development?

Story tests are requirements that are specified in a testable form that can either pass or fail. The idea of using story tests to communicate requirements has been around for many years in the Agile community, but it is having problems being adopted by practitioners unlike other Agile methods. Tools such as Fit [Fit11]] were developed, but the community is still unsure how story tests should be implemented in real life situations. In order to suggest how story tests should be written, we need to collect more information on how practitioners wrote their Story Tests and find out what were some of the problems that they encountered.

Research Question 2: Investigate the relationship between stories, teams and defects.

We need to figure out the relationship between team members and Agile development artefacts such as code, defects and stories. Story Tests are the links between stories to the codes and eventually they serve as artefacts to discover defects through the automation of story tests. We need to analyze what factors have direct correlation between the stories and defects, because these attributes may be important in the overall understanding of what story tests are.

Research Question 3: 3) What are the factors that lead to successful adoption of Story Test Driven Development?

Story Test Driven Development is the process of using story tests to drive the software development. We need to understand the core benefits of Story Test Driven Development that no other practices can provide. One of the best ways to understand the uses of Story Tests is to observe how real life Agile teams adopted Story Test Driven Development and analyze the motivation of using Story Tests in their development process. We also need to compare and contrast different Agile teams and see how the differences in their organizations and processes led to the overall success of practicing Story Test Driven Development.

1.6 Organization of the Dissertation

We organized the Chapters as follows. We present the literature survey in Chapter 2. In Chapter 3, we discuss the research design and research methods. In Chapter 4, we provide the survey done on the issues in Story Test Driven Development. We analyzed the Agile community's response on what they view to be the goals, problems and their visions for STDD. In Chapter 5, we present a quantitative case study on how requirements can be traced all the way to the defects and whether the social networks of the organization has an influence in the defects trace. In Chapter 6 and 7, we offer two qualitative case studies of companies who practiced Story Test Driven Development. In In Chapter 8, we synthesize the findings. In Chapter 9, we conclude the dissertation with final thoughts.

1.7 Contribution to the Academic Body of Knowledge

The main contribution of this research is to approach story testing using examples and think of Story Test Driven Development as a knowledge building process rather than a software testing process. To elaborate, the knowledge building process means transferring the domain knowledge from the business experts to the developers through a series of domain examples that can be tested against code. The test ensures that the developers acquired the necessary knowledge to implement the code correctly and validate their implementation against the customer-specified tests. The examples are collected iteratively as the development progresses. Therefore, the repository of examples will grow as the development progresses iteratively and the collection of these examples will grow over time. This process of collecting, testing and communicating their requirements iteratively through example-driven story tests is what we refer to as the knowledge building process.

What do we mean by example-driven story tests? We identified that the existing interpretation of story tests tend to write the story tests like test cases – a set of tasks that can either pass or fail. This view of story tests bias the test specifications toward software developers and software testers and these tests look like unit tests (but just using different tools). The problem with the developer/tester-centric view of story tests is that unit tests and story tests eventually serve similar functions. If there is a lot of overlap between unit tests and story tests, the team would lose the need to maintain story tests. Therefore, it is better if the customers provide the test values using examples right out of their domain using the formats and tools of the domain. The developers can extract values from these

examples to setup their automated tests. This is what we mean by example-driven story tests.

Story tests are communication tool between customers and developers, not a testing tool. From customer's point of view, story tests are their way of communicating their knowledge. We identified that customers are much better at communicating their story tests using examples from their domain, instead of trying to communicate them using software testing tools. In addition, these tests should not be organized like unit tests, which tend to be organized based on how the code is organized. Story tests need to be organized based on how the customers see to be their problems. Therefore, the process of collecting these examples turns into a knowledge building process rather than a software testing process.

In this dissertation, we present four case studies that suggest that story testing is a way of communicating domain knowledge and a story test needs to be written in the formats and tools of the domain. Story testing is an Agile requirement engineering practice and not a software testing practice. We identified that the main hindrance of adopting Story Test Driven Development is that the customers need to learn the tools of the technology. Compared to the challenges of getting the customers to write the story tests, extracting data out of the examples to write automated tests is relatively easy task for developers. By using the examples of the domain as the story tests, we not only improve the communication between customers and developers, but also communicate the business context in which these software functionalities need to work. We also identified that there needs to be rewards and motivations for the customers to contribute these story test. One of the best ways to do so is if there is a community of contributors

who want to build a knowledge repository of examples. It not only solves the story testing issues, but it can also serve as documentation for software.

CHAPTER 2: LITERATURE SURVEY

In this Chapter, we present a literature survey on Story Tests and Story Test Driven Development and related background for our research. We start by providing an introduction to requirements engineering from an Agile software engineering perspective, followed by a literature survey on Story Test Driven Development. We also give an overview of the principles behind Agile development and how these are reflected in the team organization and project management. We explain the differences in terminologies between traditional software engineering and Agile software engineering.

We provide a literature overview of the papers published in the area of Story Test Driven Development and categorize them into different views. We think it is important to see how each view is different even within the Agile community and we think the categorization will highlight the different problems that Story Test Driven Development needs to solve. In addition, the categorization will highlight the different approaches within the community.

2.1 Theory of Management

Agile software engineering is built on a different philosophy than traditional software engineering. It is inherently impossible to interpret and analyze Agile teams by directly comparing them with traditional software engineering teams. In order to figure out how Story Tests fit into the Agile software development process, we need to first figure out the fundamental philosophical differences that guide these two software engineering methodologies.

There are two types of management theories that mainly influence the Software Engineering methodologies: Taylorism and Lean Production. The difference between traditional and Agile software engineering methodologies is actually an argument about these two different management theories. The traditional software engineering methodologies are based on Taylorism and Agile software engineering methodologies are based on Lean Production.

2.1.1 Taylorism

Frederick Taylor produced a theory of management called *Scientific Management*, otherwise known as *Taylorism.* It is much better known as Fordism after its successful adaptation in the automobile industry by Henry Ford. The principle behind Taylorism is to apply 'scientific methods' to improve efficiency, mainly through labour productivity, which led to mass production. In his publication, *The Principles of Scientific Management* [T11] published in 1911, he states that the solution is not in finding extraordinary people, but in managing the inefficiency that lies in the lack of systematic management. If his principles are applied correctly, he assured that extraordinary results can be achieved even with people with little or no skills. He argued that his management philosophy can be applied by any organization.

Taylor believed that the vast majority of workers are incapable of management [M03] and managers did not have enough control over the production process. He believed that their lack of control was the main cause of the inefficiency. In addition, he believed that workers have natural tendency to fool around and expect the same pay

[M03]. Therefore, Taylor proposed a scientific approach to managing the processes and workers.

The first aspect to Taylorism *is interchangeable people*. The organization needs to find work that the worker is naturally good at and maximize his/her abilities by making him/her focus on that single task only. In addition, the supervisor must provide each worker with training and assessment based on how well the worker does that specific task. In turn, everyone can be replaced with another person at any time, because the knowledge is embedded in the process, not in the individuals. The second aspect is to replace the 'rule of thumb' work methods with scientific approaches that measure the efficiency based on what is produced and observed. The assumption in Scientific Management is that there is a single "best way" to do a job, because the best method is the one that optimizes the assessment metrics. The third aspect is to divide the workers into managers and workers, such that managers can plan the work and the workers perform the tasks as they are planned. In this way, the managers can focus on planning only, rather than be overwhelmed with both the tasks of producing and planning.

In 1912, a year after his publication, the Congressional committee invited him to defend his theory, suggesting that his approach is dehumanizing. However, the success of his methodology was hard to argue. The application of his methodology in labour specialization and mass production was a huge success, especially how his theory was applied successfully in Ford Automobiles. Even though Fordism became much more well known to most people due to his fame, Taylorism is the actual underlying theory. Rifkin suggests that "Taylor has probably had a greater effect on the private and public lives of men and women of the twentieth century than any other single individual"[K97]. By

1950s, and even most people who grew up in our times, would not second guess whether there is even any other way of managing people than Taylorism as it is engraved in our society in how we manage people and processes. In addition, it is hard to argue against Taylorism because how could anyone argue against optimization for efficiency using scientific approaches.

However, Taylorism has problems in modern day organizations, especially in software development management. Taylor used a reductionist approach, which decomposes the production into discrete processes and tries to optimize each discrete process. However, individual optimization of parts may not always lead to the overall efficiency as it sometimes led to overproduction of one part only and thus leading to wasted resources. It assumes that the problem is static and there is only one best solution to each part of the problem. It worked in the manufacturing industry at Taylor's time when there was little competition and the business environments did not change quickly. The tools and technologies stayed longer and did not become obsolete in a few years. In short, Taylorism assumes that 1) the problem is predictable; 2) the problem is controllable and 3) the focus should be on optimization [M03]. When we mention traditional methods in software engineering, we are referring to methodologies that are based on Taylorism. An example of the Tayloristic approach would be the Waterfall model [B83].

2.1.2 Lean Production

The other management theory is now known as Lean Production methods or Just-in-time approach. The history of Lean management begins in 1927 at Toyoda Automatic

Looms, which manufactured automatic power looms. However, the machine was complex and difficult to maintain without very highly skilled weavers. Toyoda decided to invite an American engineer, Charles Francis, to help him manufacture his looms and Francis introduced *interchangeable parts* to the manufacturing process. Due to the complexity of the machine design, there was no room for interchangeable people [PP06]. The machines required highly skilled weavers to keep the machines running and even more highly skilled people to build and maintain the machines. Therefore, Toyoda only hired the most capable engineers to work on his looms and focused on recruiting skilled workers who can produce these complex parts.

In 1936, the company decided to get into the automotive business. To do so, the owner toured Detroit to learn how to build cars. However, he quickly realized that it was impossible for him to duplicate the mass production model for his company. His company did not have the resources to mass produce thousands of identical parts for it to be economical. Taiichi Ohno, a machine shop owner at the plant, learned about Ford's production system, but he was rather fascinated with the American supermarkets inventory system. He noticed that the shelves were always filled just-in-time before it was completely empty. In 1978, he published Toyota Production System [O78], which was based on the principle of elimination of waste and 'autonomation'. According to Ohno, 'autonomation' means automation with people. All work will stop even when slight abnormalities are detected. All workers will converge to fix the problem and the assembly line will resume only when the problem is solved. It is otherwise known as 'stop-the-line' or 'zero-inspection' approach [PP06]. It means there is no sole inspector at the end of the line who is specifically tasked to find mistakes. Rather everyone is always

looking out for mistakes and solving the problem instantly as they are found. Therefore, the system will fix mistakes before it reaches the end. In addition, the waste is identified as the weakest point in the system that needs to be improved. The assembly line is only as good as the slowest moving part. Instead of trying to optimize every part of the system individually like in Taylorism, Ohno wanted to improve one weakest point at a time. In 1990, the book *The Machine that Changed the World* gave another name for the approach, *Lean Production* [WJR90], otherwise known as just-in-time production.

2.1.3 Comparison of Two Management Theories

If you compare the two approaches between Taylorism and Lean, the Lean approach values people. Unlike Tayloristic approaches that emphasized interchangeable people, the Lean approach emphasized interchangeable parts. In Tayloristic approaches, the efficiency is obtained by optimizing the individual parts of the process, but the Lean approach focuses on eliminating wastes. While Taylorism focused on specialization of labour, the Lean production implements the zero-inspection approach, which gave all workers the power to check for the quality and stop the production at any time instead of waiting for their managers. The fundamental difference is that Taylorism assumed that people are not skilled and need to be given specific instructions on how to do their job. Lean production assumed that people are highly skilled and everyone is capable of producing quality work as well as detecting and problem-solving on their own. The lean production approach is what Agile methods are based on.These management theories originated from the manufacturing industry. However, over time, these two opposing views of management were adopted in all industries.

The literature suggests that in order to develop a practice that belongs to Agile software engineering, we need to look for three key aspects: People-oriented approach rather than an interchangeable people approach, focusing on eliminating waste instead of optimizing efficiency, people are highly skilled that they will solve the problem on their own without the centralized management. In the next section, we will describe the different interpretations of how Lean principles are applied in Agile software engineering.

2.2 Requirements Engineering from the Agile Perspective

Requirements engineering is not a distinct phase in Agile software engineering. Instead, requirements engineering is embedded in the overall iterative development process. The chronological view that requirements must begin at the start and the development must end with testing is a Traditional perspective of software development. Story Test Driven Development is a requirements engineering approach in Agile software engineering. Therefore, we need to discuss what requirements engineering is from Agile perspectives.

Highsmith states that “agility isn’t a one-shot deal that can be checked off the organizational initiative list”[H02]. Rather, “agility is a way of life, a constantly emerging and changing response to business turbulence.”[H02]. Despite some critics, “agile organizations still plan; they just understand the limits of planning” [H02]. Anyone who has worked on a real life software development project would know that plans are rarely realized exactly. It is not because the people in the team are incapable or lack discipline. Real life projects are always faced with unforeseen and unpredictable

problems. If the team is dealing with a very large project, a large team, or an experimental project with a lot of cutting-edge technologies, the team will more or less hit some unknown barriers that cannot be planned ahead exactly. It is inherently impossible for individual companies to predict how the economy will be in two years ahead, what kind of competition they will face in a year or what will happen to their employees in the next month. Therefore, Agile means planning for and reacting to the changes.

The Traditional software engineering literature suggests that requirements-related defects are a very costly problem to fix. According to Fairley's estimation, the cost of fixing requirements defects may rise by 20 to 50 times if the defects are fixed in the later stage of the development [F85]. Boehm and Basili put that number as high as 100 times [BB01]. Up to 85% of the defects are estimated to come from the requirements [HF01]. Literature states that requirements changes or introducing new requirements increase the defect rate to about 50% [J97]. However, Agile development works within the environment where requirements are constantly changing and it is meant to be applied in such development projects. In an Agile development environment, preventing requirements change is not the solution, but rather adopting to the changes. Therefore, the challenges associated with requirements engineering create very complex issues for Agile software engineering research.

In Agile software engineering, the separation of different stages of the software development lifecycle is also blurred. There are no definite phases for requirements engineerin, coding and testing. Rather, they are combined within an iteration. Agile methods are based on iterations more than phases of software development lifecycles.

Therefore, discussing requirements engineering as a separate topic in Agile software engineering is often difficult and may even be impossible.

2.2.1 Different Agile Methodologies

As mentioned earlier, the Agile methodology is made up of many methodologies, which came about separately in mid to late 90s and early 2000s. They all shared common principles [A11]. Kent Beck published *Extreme Programming* in 1999 [B99]. Schwaber published *Agile Project Management with Scrum* in 2004 [S01, S04]. Cockburn published his Crystal Clear methodology in 2004 [C04] and Poppendieck et al. published Lean Software Development in 2003 [PP03]. Dynamic Software Development Method Consortium produced their methodology called DSDM [D11]. Feature-Driven Development was published in 2002 although the concept was devised by Jeff de Luca in 1997 [PF02]. Ambler's Agile Unified Process is published in 2011 [L11], which is a modification of IBM Rational Unified Process [I11] to fit Agile principles. However, by far, the most popular methodologies among them are Extreme Programming, Scrum and Lean Development. However, recently, people do not practice these methodologies separately. It is hard to find an organization that is truly devoted to only one type of Agile methodologies. Instead, they combined the development techniques from various methodologies. For example, Agile methodologies include test-driven development [B02], retrospective meetings [DL06], continuous integration [DMG07], user stories [C04], scrum meetings [S01, S04] and code refactoring [F99].

2.2.2 Agile Team Organizations

In order to discuss who is responsible for requirements engineering, we need to discuss the roles of the people who are involved in the software development. In Agile software engineering, the stakeholders are roughly divided into ‘customers’ and ‘developers’. There are no strict guidelines on who falls into what category, but generally developers deal with the technical side of software engineering and customers deal with the business side of the development. However, it does not mean there is no division of roles in Agile software engineering.

2.2.2.1 Team Organization in Extreme Programming

In Extreme Programming, Beck divided the team into 10 roles: testers, interaction designers, architects, project managers, product managers, executives, technical writers, users, programmers and human resources [B04]. However, Beck states that XP team is not fixed and rigid. The goal is to have everyone contribute to the success in whatever form they can. In addition, there is no one-to-one mapping from a person to a role. Project managers can work on architectures and programmers can create stories – if that made most sense for the team at the time and they have the skills to do so. Or a programmer can both code and architect if they have the skills and knowledge to do so.

2.2.2.2 Team Organization in Scrum

In Scrum, the stakeholders are divided into three core scrum roles. The core scrum roles are the product owner, the team and a Scrum Master [P10]. Schwaber defines the product owner as “the one and only person responsible for managing the Product

Backlog and ensuring the value of the work the team performs" and "maintains the product backlog and ensures that it is visible to everyone" [S09]. This is the person who speaks for the customers and prioritizes the to-do list, so that the functionalities with the most business value get implemented. Sometimes a person may be a product owner and a Scrum Master.

Scrum Master is the person who is responsible for removing any obstacles that get in the way of delivering the software at the end of the sprint (which is the term used in Scrum methodology in lieu of iteration) [P10]. This is the person who acts as the buffer between the developers and any outside influences and also enforces the rules of Scrum.

The Team is the rest of the people who actually develop the software product. They are everyone who analyze, design, develop, test and document the software product. The team is self-organizing. In addition, people who do not belong to any of the three roles above are called stakeholders. They may be the customers who will pay for the software product at the end or the actual end users. Or they may be sponsors for the project.

Scrum likes to use the chicken and the pig analogy to explain their role division. In the story of the chicken and the pig, a chicken and a pig are trying to open a restaurant that serves ham and eggs. In such situation, the pigs are committed, but the chickens are merely involved [K10]. The pigs are the members of a Scrum team who are committed to the work in the sprint and chickens are the customers and stakeholders who do not have the personal commitment to the work. Chickens can influence the project direction, but pigs need to commit to implementing the features. As such, the chickens, namely the customers, cannot change their goals and interfere with the development within the time

period of a Sprint. A Sprint is usually defined as a period of two weeks to a month. Therefore, in Scrum, the division between the developers and customers are their level of personal commitment to the project.

2.2.2.3 Team Organization in Lean Software Development

However, Lean Software Development does not believe in just 'self-organizing teams' [PP09] – a manager is still required. Poppendieck writes, "when the work system is the problem and the manager has little understanding of how it works or why it is not working, self-organizing teams may help, but only if they have the skills to see and solve the problem in the work system" and adds, "this may happen with mature teams, but certainly not with every team" [PP09]. Lean Development states that line managers are needed, but they need to have good knowledge of the work they manage and understand how work should get done. However, Poppendieck writes, "managers do not focus on achieving goals and they do not tell people what to do", but rather they "focus on improving the system whereby the organization's work gets done" [PP09]. Lean Software Development argues that there is a need for a manager and workers, but the manager's job is about "helping everyone learn how to see problems, solve problems and spread the knowledge" [PP09].

Schmidt and Lyle compared Lean teams like a Jazz band. A musician has a mastery of his/her instrument, but also must work within the team. It is important to have an empowered team that works in synergy and synchronicity and develop leadership for all team members [S10]. In addition, the methodology recommends that team members be rotated into the customer's shoes. A developer who does not have the business

perspective cannot develop good software. It is not enough to just be told what the customer needs. It is more important to have “some sort of intuitive, common sense grasp of what the customer might want, although this must never be completely substituted for constant interaction with and feedback from real customers” [S10]. Second, the team members need to grow their skills constantly. Poppendieck states, “deskilling workers creates interchangeable people while upskilling workers creates thinking people” [PP06]. Therefore, Lean Software Development avoids explicit partitioning of the team into specific roles. Rather, it is a task that each team needs to figure out on their own.

Agile methods have a lot of different interpretations on how each Agile team decides and divides their work. There are a lot of variations on how one can go about organizing their teams. However, all of the methodologies agree that constant interaction with the customers is important in order to make sure that the software development is going in the right direction.

2.2.3 Requirements Artefacts

Agile methods communicate requirements through tasks that are broken down into smaller workable pieces. All of the Agile methods agree that these tasks need to be broken down in consultation with the customers and the developers. However, there are also differences in how these tasks are composed. Most literature uses the term *story* or *user stories* to refer to these tasks.

2.2.3.1 Stories in Extreme Programming

In Extreme Programming, stories are defined as “plans using units of customer-visible functionality” and small enough to estimate the development effort [B04]. He observed that usually even just implementing 5% of the requirements would provide all of the business benefits of the whole system; the rest of the ‘requirements’ are just nice-to-have’s [B04]. Requirements that are not estimated or prioritized are not useful. Therefore, the main difference between stories and requirements is that stories are requirements that are broken down in a way that can be estimated. Beck states that “estimation gives the business and technical perspectives a chance to interact ”[B03] and prioritize ideas that have most potential for business value and technical feasibility. Beck states that “when the team knows the cost of features it can split, combine, or extend scope based on what it knows about the features’ value” [B04]. Therefore, unlike requirements that are instructions that are handed down by the customers, stories are tasks that can be estimated and evaluated for both technical feasibility and business value by both customer and developers. Because the estimation and technical feasibility is coming from the customers and the developers, both parties have come to conclusion about what is really possible and arrive at more realistic outlook on their development plans. There is no specific guideline on how these stories need to be broken down, but they must be small enough that developers can estimate in terms of a hours and non-technical enough for customers to understand what the stories are about [C04]

2.2.3.2 Stories in Scrum

Scrum also communicates using stories. The requirements are broken down into stories and they are put into *Product Backlogs*. Schwaber uses the term Product Backlog items instead of stories in his books. He defines Product Backlog items as "a prioritized list of functional and non-functional requirements and features to be added to an existing product" and "are granular enough to be readily understood by the Scrum Team and developed into an increment within a Sprint"[S07]. In each *Sprint*, which lasts two weeks to a month, all stakeholders gather to figure out which product backlog items need to be implemented. Mike Cohn's explanation of Scrum uses the term, user stories, instead of product backlog items [C09]. Cohn defines user stories as "a short, simple description of feature told from the perspective of the person who desires the new capability, usually a user or customer of the system" and "are often written on index cards or sticky notes, stories in a shoe box, and arranged on walls or tables to facilitate planning and discussion" [C09]. The most important aspect of Cohn's definition of the user stories for Scrum is that they are conversation starters, not the detailed documentation for the requirements. He recommends the following template for writing the user stories: "In order to <achieve value>, as <type of user>, we want <some goal>" [C09]. He said the most common mistakes that people make who are new to Scrum is that they try to write everything down on the index cards and they are only written by a subset of business analysts. These are often mistakes from people trying to bring their practices from traditional software engineering into Agile methods. Conversation is more important than documents in Agile methods.

2.2.3.3 Stories in Lean Software Development

Lean software development also uses the term, stories, to denote requirements. Stories are “units of development that can be estimated reliably and completed within a few days” [PP06]. Lean software development also uses the term, product backlog, which is defined as “a prioritized list of desirable features described at high level” [PP06]. The main difference between Scrum and Lean Development for the product backlog is that Lean Development considers backlog items as “large-grain bullet points”, or “epics” rather than “stories”, because it believes that “detailed analysis must be delayed until the last responsible moment”[PP06]. Instead of the term, Sprint as in Scrum, Lean Development uses the term, Iteration. Therefore, in each iteration, these backlog items are broken down into more manageable stories. The stories are “analyzed by team members who understand the customer domain and the technology” [PP06], which emphasizes that generating the stories are a team effort. Poppendieck states that “a good story is a well-defined unit of implementer work, small enough so that it can be reliably estimated and completed within the next iteration”[PP06].

In summary, Agile methods like to break down requirements into stories. Stories are conversation starters or reminders, rather than an artefact that contains complete requirements information. Stories need to be written in such a way that both developers and customers can understand and estimate them and both parties can write new stories. Agile methods are meant to be applied in a development environment where requirements are constantly changing, thus stories are meant to facilitate such transient business context.

2.2.3.4 Requirements Artefacts in Traditional Software Engineering

We have been frequently challenged by others for these definitions of requirements and stories, as these definitions are quite different from the definitions used in traditional software engineering. First, the definition of requirements is much more complex in traditional software engineering. Sommerville and Sawyer defined requirements as [SS97]:

> *A specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be constraint on the development process of the system.*

IEEE Standard Glossary of Software Engineering Terminology defines a requirement as [I90]:

1. *A condition or capability needed by a user to solve a problem or achieve an objective.*
2. *A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document.*
3. *A documented representation of a condition or capability as in 1 or 2*

In traditional software engineering, requirements engineering has levels of requirements: business requirements, user requirements and functional and non-functional requirements [W03]. The term, *system requirements*, refer to "the software functionality that the developers must build into the product to enable users to accomplish their tasks, thereby satisfying the business requirements" [W03]. Requirements do not include design information, implementation details, project planning information or testing information

[L00]. Traditional software engineering also defines how these different requirements must be gathered and written down. Compared to these processes, Agile doesn't have specific definitions of requirements such as these. The term, requirements, are used more or less to describe the customer's wishlist. As mentioned above, these differences arise from fundamental differences in the management theory: processes over people. Agile does not believe in compartmentalizing the requirements engineering into processes or phases. It simply starts its development from what the customers wanted and narrows down the requirements through conversations and development iteratively.

In addition, the term, stories, as used in Agile methods seem to confuse people to think that it is same as use cases and stories in Rational Unified Process (RUP). RUP uses the term, *Storyboard*, which is a way communicating a specific story "to understand the overall flow and interactions" and "conception description of system functionality for a specific scenario"[SK07]. In RUP, the term story refers to a specific scenario – one possible narration of how the system could be used. This scenario can be communicated through a use case and RUP also has a specific guideline on how it should be written. In Agile, the term, story, should be understood more as the start of a conversation, not as one possible scenario of how system could be used.

Due to the underlying philosophical differences between traditional and Agile methods, people coming from traditional approaches may find Agile approaches somewhat lacking in details and oversimplified, especially in terms of processes. But this is the fundamental difference: traditional methods value processes and Agile methods value interactions.

However, how does one know whether the stories are communicated clearly? In traditional methods, many layers of guidelines and processes ensure that as many errors are found in the requirements stage as possible. However, such processes are not in place for Agile methods. Thus, there has been talk of Story Test Driven Development. In addition to the stories, STDD is a way to ensure that the requirements are implemented correctly using test-driven development method.

2.3 Fit

Before we discuss the existing literature on story tests, we need a brief introduction to the conventional way of doing story testing. The original tool that started story testing (or acceptance testing) is Fit [Fit11]. This is the tool that inspired other tool development and the discussion on what Story Test Driven Development is. Cunningham and Cunningham who maintain the tool and the site state:

> *Fit is a tool for enhancing collaboration in software development. It's an invaluable way to collaborate on complicated problems – and them right- early in development. Fit allows customers, testers, and programmers to learn what their software should do and what it does do. It automatically compares customers' expectations to actual results.*

Here is a diagram of how Fit specification looks like as taken from their website [Fit11] (Figure 1).

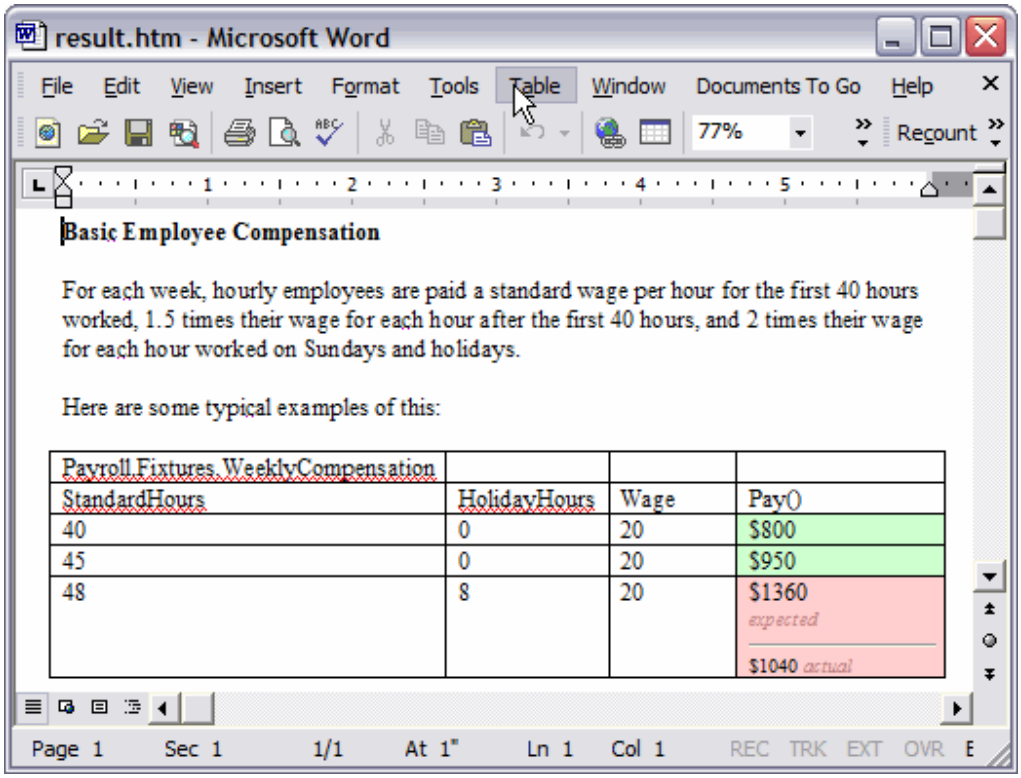


Figure 1: A Fit document showing how the tests are specified. The green cell means the test passed .The red cell means the test failed.

The table contains examples of how the function needs to be written. The first row tells what kind of format is being used to write the table. The second row gives headers. The remaining rows provide example values. The developers would write fixtures using Fit framework. The fixture tells Fit how to extract data and use them to write the acceptance test (story test). Once you execute the test that is written using Fit fixtures, the test would return a result (such as one shown in Figure 1) highlighting the output columns with either green or red. The green cell means the output value from the test corresponds to the expected value that is specified in the specification. The red cell means the output value from the test did not correspond to the expected value.

The main problem with Fit is that it is an incredibly useful tool for the developers in terms of making the test automation easy, but it is not easy to write the example specifications because it needs to follow strict rules on how these examples can be specified. As we will present in this dissertation, many examples in different domains cannot be specified according to the strict rules of how Fit tables need to be specified.

2.4 Story Tests

The first problem is in the terminology that is used for Story Test Driven Development. This section will begin with the definitions of story tests as defined in different agile approaches. We list the definitions here to sort out what people determined to be the scope of story tests, because different papers provide different characteristics about story tests.

2.4.1 Story Tests from Business Perspectives

The first category of definitions of story tests refers to business aspects of the specification. For example, Read et al. states “acceptance tests are high level tests of *business operations* and are not meant to test internals or technical elements of the code, but rather are used to ensure that software meets *business goals*” [RMM05] and “acceptance testing is a formal technique to ensure that a system satisfies the *expectations* of the customer who commissioned the software. Acceptance tests verify code against the requirements and act as a type of check of contractual obligation between customer and developer” [RMM05]. Read et al. also states that “these tests are written from the *perspective of the user*, and test the system as a whole (as opposed to unit testing, which

tests technical detail)" [RMM05]. Melnik et al. state "acceptance testing must proceed from the user's perspective (not the developer's)" [MM05].

The story tests describe the functionalities from business perspectives. For example, Read et al. state "the motivation for acceptance testing is to *demonstrate* working functionality rather than to find faults (although faults may be found as a result of acceptance testing" [RMM05b]. It also states that "they are traditionally specified using *scenarios* and performed by quality assurance teams together with the user or representatives (eg. Business analysts). [RMM05b]

Story tests are also to be written by the customers. "Acceptance test are different from unit tests in that the latter (unit tests) are modeled and written by the developer, while the former is at least modeled and possibly even *written by the customer*." [MRM04]

2.4.2 Story Tests as Examples

Some definitions state that story tests are about providing examples. Mugridge states that "in Executable Acceptance Test Driven Development, customers write Executable Acceptance Tests – executable, *business oriented examples* – for each scheduled story. The goal is to encourage clear communication of essential *business needs* (and ways of meeting those needs) using concrete examples." [M08] and "executable acceptance tests *evolve* through collaboration and thus clarify the domain and scope for all project participants, enabling conversations that build shared understanding among team members" [M08].

2.4.3 Story Tests as a Project Management Tool

Story tests are also used to report the project progress to the customers. For example, Melnik et al state that “acceptance testing is conducted (preferably by the customer) to determine whether or not a system satisfies its acceptance criteria. The objective is to provide *confidence* that the delivered system meets the business needs of the customer” [MM07]. “Acceptance test-driven development is a software development methodology that emphasizes acceptance tests as a *main project artifact*. These tests are used both to represent software requirements and business rules and to guide software development via frequent test runs.” [SN08]

2.4.4 Story Tests as a Quality Assurance Tool

Story tests are also used for quality assurance. “Functional tests no longer merely assess quality; their purpose now is to *drive quality*”[A07]. Sauve et al states that “acceptance testing is a *validation activity*, performed by the customer, on the entire system, just before the system is delivered and aimed at judging if the software is acceptable” [SN08]. Mugridge states that “executing Executable Acceptance Tests as automated tests can help developers determine when new functionality is complete as well as if *any existing functionality has been broken*” [M08]

Frequent testing can prevent accumulation of defects before they become impossible to fix. However, software testing is double-checking. There needs to be a method of preventing defects, which is where Test Driven Development comes in. 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.

2.4.5 Story Tests from Different Perspectives

Kerievsky describes story tests from more of developer and testing perspective [K11]. Kerievsky describes story testing 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 the domain knowledge, but turning them into tests requires testing knowledge. Therefore, story testing may require domain experts/subject matter experts and quality assurance experts. Story testing involves identifying the *minimal tests* that will cover all boundary conditions.

Unlike Kerievsky who states that minimal boundary values must be tested, Marick suggests that these tests are for exploration [M11]. Therefore, he likes to call story testing as *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 (not so much about covering test values). Getting the tests precisely right isn't the point in the beginning, because coming up with tests may require more understanding. The Executable Acceptance tests evolve with better understanding as the implementation begins.

Fowler likes to call this process, ‘Specification by Example’ [Fo11]. 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 (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”.

For people who approach from more traditional software engineering perspective, story tests may go by the name, *scenario tests* [K03]. Kaner states that “a scenario is a hypothetical story used to help think through a complex problem or system”. As the name suggests, scenario tests are tests based on scenarios. However, unlike traditional testing practices, Kaner states that “scenarios [in his perspective] are meant to help you learn the product”.

In more traditional testing practices, testers are given a checklist to test, because the belief is that the best way to learn software for testers is to run software “keystroke by keystroke”. However, Kaner found that testers actually find more defects and learn the software better if they are given a set of scenarios to investigate. Scenarios can also turn into documentation about software. Scenarios are also good for identifying the experts, because they will use software differently as they gain experience with software. Scenarios are also good for “surfacing requirements-related controversies” because

people bring in different views about solving the scenarios. Kaner also states that good scenario tests must be motivating and credible.

Crispin and Gregory used the word, “Agile Testing” instead – to mean customer facing and business facing tests. They state that testing in agile software engineering is different from traditional testing in that everyone is involved in testing [CG09]. The teams are divided into a customer team and a development team only. The business team includes business experts, product owners, domain experts, product mangers, business analysts and anyone who is on the business side of the project. The developers write, design and maintain the automated tests and the code. The testers belong to both groups because they help customers write tests and help developers maintain quality. Crispin and Gregory also state that the purpose of the business facing tests (or story tests) are to help elicit examples and context for each story, so that these tests can help guide programmers as they write the code. Crispin and Gregory also like to use the term *coaching test* because these tests can help developers understand the domain. Once the examples are acquired, tests can be created from the examples. The tests are an executable format of the examples.

To summarize, one of the key viewpoints that keep appearing in most of the literature is that the purpose of story testing is to *understand* and *explore* the domain. Most of them emphasized that story tests help people *learn* about people’s expectations, business priorities and the domain knowledge. Thus, these three keywords, understand, explore and learn about the domain, form the key motivations for the research topic.

Story tests examine the business operations, business goals and business needs. Story tests ensure that the system fulfills customer’s expectations and the tests need to be

written from the customers' perspective. The tests provide business oriented examples. The purpose of automating story tests is partly to serve as a project management tool, particularly to report the progress of the development. The story tests are also used for demonstrating the working functionality. Finally, the definitions suggest that Story Test Driven Development is a validation activity. However, there are still many variations on what story testing is and these claims still need to be backed with empirical evidences.

2.5 Literature Survey of Story Test Driven Development[1]

As shown in chapter 2.3, there are many interpretations on what Story Tests are. We needed to look the knowledge gap from the existing literature and categorize them into what is known and analyze different views on story tests and Story Test Driven Development. We collected papers (to the best of our effort) related to story-test driven development that are published in peer-reviewed conference proceedings, magazines and journals from 2001 to 2010. We collected 49 lessons learned papers, 8 tool development papers and 8 research papers. 2001 had 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

[1] This section appeared in the following paper: Park, S., Maurer, F., A Literature Survey on Story Test Driven Development, Proc. Of 11th International Conference on Agile Processes and eXtreme Programming, Trondheim, Norway, 2010. The copyright release form is attached in Appendix II.

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.

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)[G01][DD08]. 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?

We only included papers written in English. We categorized all of the literatures published that are related to Story Test Driven Development into 7 themes: cost, time, people, code design, testing tools, what to test, and test automation issues. 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 other research collaborators. After describing the points from the lessons learned and tool development papers, we also describe the findings from the research papers. We describe whether the research papers support the points described in the lessons learned papers. We also describe whether the research paper supports the points described in the lessons learned papers.

2.5.1 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 [F01, SSO05, HH08] 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 [F01]. The team’s continual small

adjustments (on time) keep the project on course on time and on budget"[F01]. Schwartz also states that the automated story tests can "run often and facilitate regression testing at low cost"[SSO05].

Four papers [F01, SSO05, CH01, CHW01] 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."[CH01]. In addition, four papers stated that their teams did not have the budget necessary to automate the tests [CH01, CHW01, A04, A07]. 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."[A04].

There were no research papers that explicitly analyzed the cost and budget aspect of STDD process or the tools. However, we assume that given an appropriate tools and practice, the benefit of STDD will outweigh the cost.

2.5.2 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 [F01, HH08, CH01, CHW01, A07, WL04, R04, M08, TD09, HH09, St09, KPGM09, MM05]; 2) adapt to requirements changes with the help of instant feedback, which can help keep the project on time [CH01, HH09, MM08]; 3) continuous

verification (test anytime, more often, repeatedly)[HH08, M08, HH09]; 4) better estimation of the stories [WL04, OP09]; 5) immediate defect fixes [St09, K06]. For example, Rogers [R04] 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 [HH09] 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 [K06] stated that STDD is “excellent for regression testing and allow for continuous integration, in turn enabling issues to be handled immediately when they appear”, which would serve similar purpose as unit testing but now the results would be meant for the customers.

However, some lessons learned papers identified three issues related to time: 1) writing and maintaining tests took considerable time [HH08, St09, KPGM09, A04, GHHW05, TKHD06], 2) it can take long time to execute the tests [R04, GHHW05, ABS03, MC05, HK06], and 3) there can be a lack of time to build the necessary testing tools and infrastructures [St09]. For example, Ghandi et al. [GHHW05] 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 [ABS03] 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 [St09]

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, [MMC06], 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, [MM07], discovered that timing was a matter of discipline more than an actual timing problem.

2.5.3 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 [HH08, M08, OP09, HK06, SP04, ARS07, GBGP07, KNR09, CD07, PM08], 2) confidence about the progress and deliverables [HH08, CHW01, St09, K06, HK06, R04, ARS07, MLSM04, ABL09], 3) better awareness for testing in the team [R03, TKHD06, MS07, HH08, St09, MM05], 4) encouragement of collaboration between right people [R04] and 5) anyone can quickly understand what’s been developed [GHHW05, TKHD06]. For example, Abath [ARS07] 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 in the software that is being developed and automatically enforced focus on the client’s interests,

preventing feature creep." Talby et al. [TKHD06] 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. [GHHW05] 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 [R04]. [R04] 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 [GHHW05, ARS07]. For example [GHHW05] 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 [HH09, ABL09, K07]. For example, [GBL+04] 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 [M08, KNR09, GBL+04, Su07, RMM05], but most of them overcame the problem quickly. For example, [M08] 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 [CHW01, TD09, TKHD06, Su07], or a separate group of dedicated developers/testers were created for STDD [CH01, GHHW05]. One team used pair story testing method [CH01]. 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 [WL04, R04, M08, HH09, MM08, ABS03, GBGP07, MT03, DWM07]. In some cases, developers wrote the story tests with the customer collaboration [A07, TD09, HH09, MMR03]. In some teams, the QAs wrote the tests in collaboration with the customer [CHW01, MLSM04].

We found seven research papers that looked into people related issues. [MRM04] performed an experiment on how quickly developers can learn to use a STDD tool. [MRM04] discovered that "FIT[MC05] 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 [RMM05] 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 [MMC06] 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 finding from [MMC06] 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. We suspect that the combined team produced more comprehensive tests because different perspectives were represented.

On the contrary, the research in [RPT+08] suggests that experienced developers gain much more benefits from Fit tables in software evolution tasks, suggesting that previous experience does matter. It suggests that existing skills do influence the amount of benefits one can get from story testing tools.

The research in [MM07] 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 [PM09] 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 [RTCT07] 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.

2.5.4 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 [F01, CH01, A04, GHHW05, HK06, PW03, M05, PM08, K07]. For example, Kongsli [K07] 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 [A07, S03, ARS07, YRG09, SNC06]. For example, [ARS07] found that "fewer bugs were discovered when the system was placed in production." 3) The STDD can drive the overall code design [HH08, M08, YRG09] and 4) developers had a better understanding of their code [ABL09]. For example, Abbattista [ABL09] 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 [St09, M08]. 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 [RMM05] discovered that more quality code is produced the first time. The research in [RPT+08] 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. As to what was written for Fit tests is not explained and no examples were provided in the paper. The researchers in [RTD+08] 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 [MMC06] 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.

2.5.5 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 [CH01, MM08, MMR03, AB04, ABV05]. However, there are clear disadvantages 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. In addition, these tools are unsuitable for gathering requirements as GUI would not be developed yet. Instead, some people use unit testing tools such as jUnit and nUnit [F01, St09, ABS03], 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 [F01, CH01, A04, ABL09]. Some people used XML for the test specification [CH01, A04, KNR09, MT03, NM05]. Some people preferred scripting

languages or API based tools such as Selenium [CH01, St09, K06, HK06, KNR09, K07, ABV05]. But most people used tabular and fixture based tools such as Fit [SSO05, HH08, S03, M08, HH09, MM08, A04, GHHW05, MC05, R04, KNR09, MLSM04, ABL09, GBL+04, Su07, MT03, DWM07, NM05, CSGM06, MS07, PM08].

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 [A07, R04, KPGM09, HK06, ABL09, CSGM06]. For example, [R04] states "customers, however, do not use an IDE; now while you can teach them to use an IDE, which is something that we 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 [GBGP07, KNR09, CSGM06, CD07]. For example, [CSGM06] integrated MatLab to work with Fit and [CD07] integrated wiki and Mantis to their STDD tool. Some people felt there was a need to integrate with distributed automation framework such as STAF [St09, KNR09]. 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 [KPGM09, ABS03, MMR03, ABV05], automatic test data generation [TKHD06, ABV05] and automatic documentation generation [R04, OP09, PW03]. Automatic test generation could mean automatically creating test fixtures from Fit tables [KPGM09] or creating tests from sample web pages by tracing user inputs from a web page [MMR03]. For automatic test data generation, [ABS03] tried to mine input data provided by the customers and feeding them into the tests automatically. Some

people argued that story tests could automatically be turned into user manuals. [PW03] 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 [KPGM09], refactoring of the tests [CH01, A07, M08, KPGM09, OM08] and test organization features [R04, M08, KPGM09]. For example, [OM08] provides an ability to automatically refactor story tests. [M08] 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, [CKM09] 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 [CMK09] 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.

2.5.6 What to Test in Story Test Driven Development

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 [S03, ARS07, MMR03, AB04, ABV05], web services, web applications

and network related issues [K06, KNR09, MT03], backend functionality (functional requirements) [OP09, ARS07], performance [CHW01], security [CHW01], stability [CHW01], non-functional requirements [OP09, ABS03], end-to-end-customer's perspective of the feature [HH08, MMR03], regression testing [F01, A07, S03, R04, St09, K06, A04, MC05, KNR09, Su07, DWM07, MMR03, MS07], user interaction [A04, R04, M08, AB04], concurrency [ARS07, KNR09], database [KNR09], only the critical features as judged by the developers [HH09], and multi-layer architecture of software design [PM08]. Finally, most people thought the purpose is not so much about testing, but to communicate the requirements with the customer in an unambiguous way [HH08, R04, M08, HH09, KPGM09, MM08, 31OP09, HK06, K07, YRG09, PM08].

No empirical evaluation of this question exists.

2.5.7 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 [A04, R04, St09, KPGM09, ABS03, MMR03]. For example, [A04] 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 [A04, A07, An04, 38]. For example, [HK06] 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. The paper did not specify how many stories were involved. Similarly, [GHHW05] 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 [CH01, A04, A07, R04], 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 [CH01, A07, M08, HK06, SP04]. One author desires for better usability of the testing tools [A04]. Some people desired for more readable test specifications [HH08, A07, R04, M08, A04, GHHW05, ARS07, GBGP07, AB04, GMS05, R03]. Some people thought keeping track of the history of the tests is important [GHHW05]. One author worried that the team ignored the tests because there were too many false alarms [HK06], 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 [HK06, ABL09]. One author argued that the problem is with the incompatibility of different platforms and languages for the tests [GBL+04]. Some people emphasized tests should be written using more reusable objects and services [GHHW05, ABL09, MS07]. 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 [CH01, K06, M05].

No research paper analyses these issues.

2.6 Analysis of the Literature Survey

The number one question that arises from the literature survey is what are story tests? The results from 2.5.5 suggest that people are using different testing tools to specify story tests, but these testing tools are unsuitable for specifying story tests. As much as there are different tools for specifying story tests, we are also noticing different uses of story tests. For example, literature in 2.5.3 suggests that people used story tests for communication with stakeholders about the development progress and deliverables, to improve awareness for testing in the team and finding the right people for collaboration. On the other hand, literature in 2.5.4 suggests that story tests are used to improve code design. There is clearly a lack of understanding on the purpose of Story Tests. Literature in 2.5.1 and 2.5.2 suggest that the main hindrance to practicing Story Test Driven Development is finding ways that will not be too costly and time consuming. We need to discover the uses of Story Tests and figure out ways in which the benefits of practicing Story Test Driven Development will outweigh the cost and time that are involved.

2.7 Summary

In this section, we presented the literature survey on management theories, Agile software development and Story Test Driven Development. In the next Chapter, we will present the research goals and research methods used to conduct our research.

CHAPTER 3: RESEARCH APPROACH

In this section, we present the research goal, research methods, research design and the evaluation criteria that we used to conduct our research. We present the research goals and the objectives for each of the case studies as well. In this Chapter, we explore research methods and explain how and why we employed those methods to conduct research. As most of our research is qualitative in nature, we go into more depth on the limitations, assumptions and the reasons how we overcame the challenges of conducting qualitative studies in software engineering.

3.1 Research Questions

The main goal for research is to investigate why people use Story Test Driven Development in Agile software development. There are two main research questions.

Research Question 1: What problems are faced by Agile teams in practicing Story Test Driven Development?

Research Question 2: Investigate the relationship between stories, teams and defects.

Research Question 3: What are the factors that lead to successful adoption of Story Test Driven Development?

Currently, there are many ideas on what Story Tests are and how to practice Story Test Driven Development, but there is no analysis on why people practice Story Test Driven Development and what works. The first goal is to gather different ideas and strategies that people came up with for Story Test Driven Development. The second goal

is to figure out what Story Test Driven Development can actually offer to Agile teams. The next goal is to observe teams that are actually practicing Story Test Driven Development and analyze our observations with another team that practiced story test development in a similar way but failed to make it work.

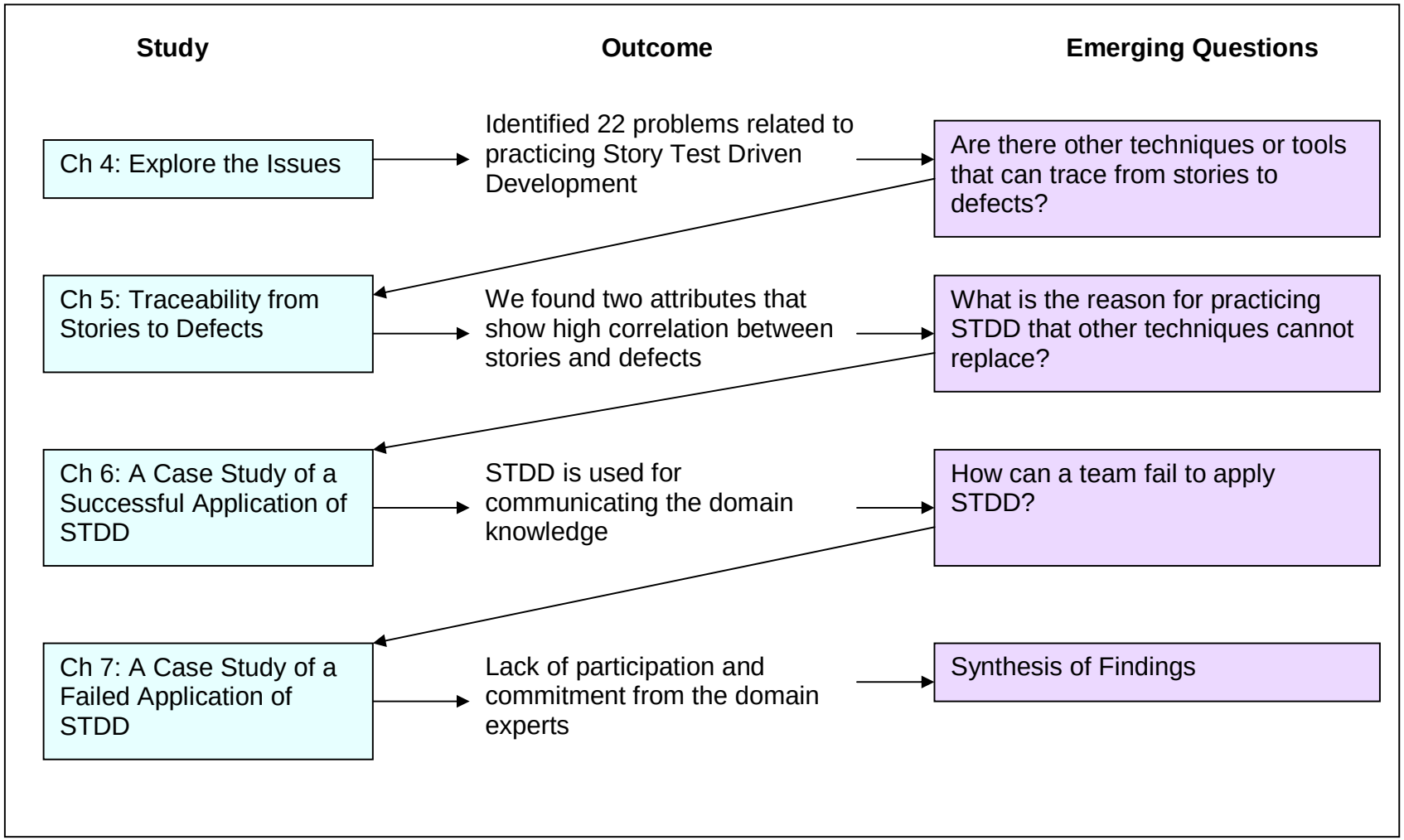
Table 1 outlines the overview of the research questions, objectives for each study and the outcomes of the study. Figure 2 outlines the overview of the outcomes and the emerging questions for each study and how those questions were used for the next study. With Table 1 and Figure 2, we are attempting to provide an overview of research and how these studies relate to each other. In chapter 4, we discovered 22 uses of story tests. The power of Story Tests comes from its ability to link from stories and codes and to the defects. Therefore, in chapter 5, we performed a case study where such tracing is possible and we discovered two attributes that have high correlation with stories and defects. We discovered that the attributes with high correlation are communication related attributes such as the number of indirect stakeholders and number of related stories. The result suggests that the occurrence of defects is highly correlated with the increase in the number of indirect stakeholders and number of related stories.

Therefore, in chapter 6 and 7, we performed case studies on why people practice Story Test Driven Development and how they used story tests. It confirms out hypothesis that stories are indeed a communication tool to discover hidden requirements and hidden complexity in finding related stories. But we also discovered that story tests are an excellent medium for transferring customer's domain knowledge to the developers.

Table 1: Research Questions and Summary of Outcomes

Research Questions Addressed	Phase	Objective	Method Used	Study	Outcome
1) What problems are faced by Agile teams in practicing Story Test Driven Development?	1	Study of existing body of knowledge	Literature review	-	• Problem statement • Initial research questions
	2	Investigate the the problems faced by practitioners in practicing Story Test Driven Development	Survey	AAFFT Forum analysis (Ch. 4)	• An initial list of problems related to practicing Story Test Driven Development
2) Investigate the relationship between stories, teams and defects	3	Investigate the traceability from stories to defects	Quantitative case study	A Large Software Development Project using Jazz (Ch. 5)	• We found two attributes that show high correlation between stories and defects.
3) What are the factors that lead to successful adoption of Story Test Driven Development?	4	Investigate the factors that lead to successful adoption of Story Test Driven Development	Observational case study	Production Accounting Software Development Team (Ch. 6)	• STDD is used to communicate domain knowledge • Use of formats of the domain for writing the story tests
	5	Investigate the factors that lead to adoption failure of Story Test Driven Development	Observational case study	Economic Reserve Analysis Software Development Team (Ch. 7)	• STDD requires a community of story test contributors

Figure 2: Summary of Studies and Emerging Research Questions



3.2 Research Methods

Unlike other science disciplines, software engineering research does not have one standard paradigm that can fit all of the research types within the software engineering discipline [S02]. One of the reasons is because software engineering has a lot of human factors and must deal with qualitative results that are often hard to compare and duplicate exactly. Therefore, there are many ways to approach software engineering research.

Our research faces two main challenges: adoption of Story Test Driven Development in real life situations and the evaluation of how the observed techniques or organizational methods worked for their practice of Story Test Driven Development. We employed several case studies. However, case studies using real life software projects have many difficulties and may not provide ideal situations for observations. As such, we can only observe how teams do their work and it is often not practical to ask the participants to change their work environments or processes just to fit our research better. Therefore, we included a detailed summary of the context in which these teams worked. In addition, the adoption usually happens gradually within the company over a long period of time while they juggle all the politics and resources. There are often a lot of human factors that cannot be controlled or even predicted in real life adoption process. In our research, the difficulty is heightened even more because Story Test Driven Development is a very new technique and not many teams are practicing it currently.

The most difficult part of pursuing research into adoption of new development techniques is the difficulty with evaluations. Because we are entering into the real life situation as an observer, it is impossible to predict what kind of results we will get or

even what kind of processes we may end up observing. Therefore, we employed qualitative observational case studies for three of four case studies.

In our research, we used empirical methods. The empirical methods are concerned with understanding and identification of relationship between different variables through observations and experimentations [WRH+00]. If an existing preconceived idea exists, then the investigator is interested in confirming whether it is true. These types of questions and their subsequent experimentations can help improve our understanding of software engineering. Therefore, the main research strategy that will be used in our research will be empirical methods.

There are two types of research paradigms for collecting research data in empirical methods. Qualitative analysis "is concerned with studying objects in their natural settings"[WRH+00]. A qualitative researcher attempts to interpret a phenomenon based on explanations that people bring to them. It attempts to analyze what the subjects in the study feel to be the cause of the phenomenon and understanding their views of the problem. Qualitative methods, especially the exploratory kind, aim to "develop pertinent hypotheses and propositions for further inquiry" [Y94] or develop of a set of 'theories' based on supporting evidence. The word, 'theories', is used in the context of a set of plausible and consistent hypothesis, not 'theories' as in unifying and undeniable force of natural law as used in math or physics.

In contrast, quantitative analysis deals with "quantifying a relationship or to compare two or more groups"[WRH+00]. Quantitative methods look for statistical significance. Quantitative research is usually performed through controlled experiments. The advantage of quantitative data is that statistical analysis can be used.

Generally, quantitative methods such as controlled experiments are appropriate for testing the “effects of treatment” and qualitative study is used to find out “why” the phenomenon occurs or to develop a hypothesis to test. The two approaches are complementary. There are three major types of strategies that can be used in empirical studies [WRH+00, R93]. In the following sections, we discuss the three methods used for the purpose of conducting our research and which studies employed these methods.

3.2.1 Survey

A survey research method is used for the study presented in Chapter 4. A survey is done in retrospective of the usage of tools or practices [P94]. Data can be obtained in both qualitative and quantitative approach either through interviews or questionnaires. The responses can be open ended or close ended. The point of the surveys is to analyze a sample that is representative of the larger population. The surveys can be used to draw descriptive, explanatory and exploratory conclusion [WRH+00, B90, R93]. However, surveys do not provide the investigator with the ability to control the execution or the measurement, but the investigator can evaluate by comparing results [WRH+00].

There are two types of surveys. The purpose of descriptive surveys is to understand characteristics or attributes about some populations. This type of surveys shows that the observed distribution of characters exists in the population, but not why it exists. The purpose of an exploratory survey is to conduct a pre-study to more thorough investigations. It is to find out what are important issues that were unforeseen at the start of the study. It is designed by providing loosely structured questionnaires to the participants. It does not start with a specific research question, but the researcher begins

with open mind about finding possibilities [WRH+00]. We employed the exploratory survey method in Chapter 4 to generate an initial set of research questions. We gathered 22 factors that people thought were the main issues involved in Story Test Driven Development. These 22 factors became the basis for further studies in subsequent Chapters. Further details about the specific research design for the survey is included in Chapter 4.

3.2.2 Case study

The case study method is employed in Chapters 5 to 7. Case studies are used for observing projects over a longer period of time on a single entity or phenomenon [WRH+00]. Data is collected with a specific purpose in mind within the observational settings. Some case studies that are quantitative in nature can use statistical methods to derive conclusions [WRH+00]. Case studies are different than experiments in that case studies are observational studies [ZW98]. The purpose of a case study is to find key factors that may have influenced the outcome [Y94, St95], but the investigator cannot isolate these factors into controls and treatments like in experimentations. Case studies are much more suitable for industrial evaluations, but their observations are harder to generalize to every situation.

We performed both qualitative as well as quantitative case studies. The benefit of quantitative case studies is that unit of analysis is usually easier to define and arriving at mathematical analysis is easier. On the other hand, the benefits of qualitative case studies are that they examine the process holistically. There is no recipe for qualitative analysis, but there are some guidelines. For example, qualitative analysis uses inductive methods

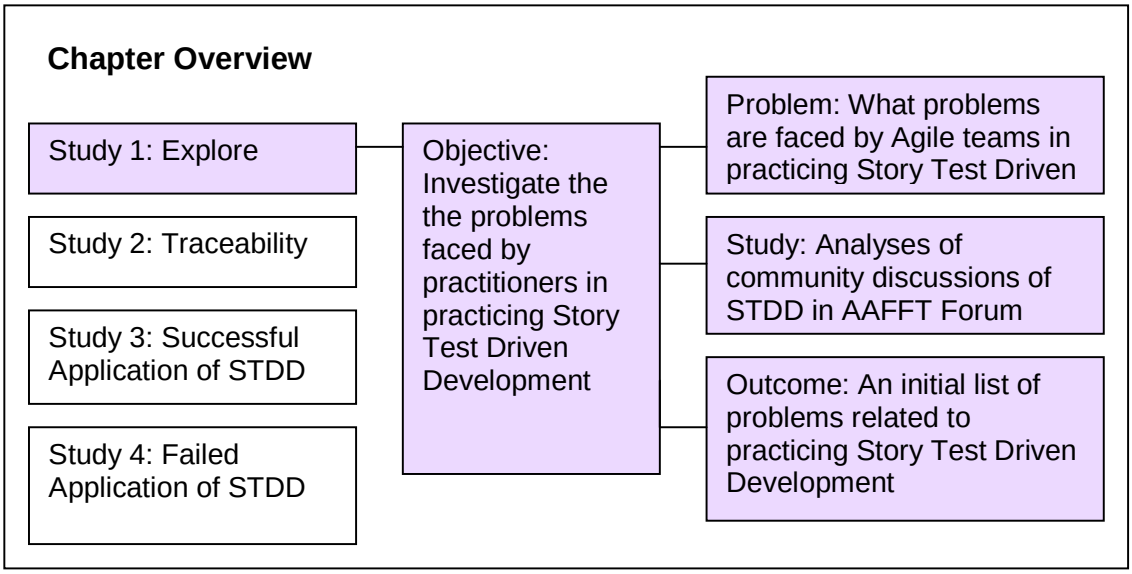
in the early stages, especially when codes are being generated for content analysis or figuring out the categories or themes. Strauss and Corbin call this process open coding [SC98]. The Grounded Theory technique by Glaser and Strauss emphasizes being grounded in data and embedding meaning and relationships from data through open coding [GS67]. Once the categories and themes emerge from data, a hypothesis can be generated. Once the hypothesis is generated from the inductive analysis, deductive analysis can follow [SC98]. Strauss and Corbin state "at the heart of the theorizing lies the interplay of making induction (deriving concepts, their properties and dimensions from data) and deductions (hypothesizing about the relationships between concepts) " [SC98]. We employed grounded theory for the study in this Chapter.

Case studies are used in Chapter 5, 6 and 7. Further details about the specific research design for these projects are included in their respective Chapters.

3.2.3 Experiment

Experiments are done in a highly controlled setting where treatments are given to random test subjects. The objective is to manipulate one or more variables and arrive at the conclusion based on statistical analysis [M97, SC88, R93]. Experiments can be used to confirm existing theories or hypothesis. They can also be used to explore relationships between specific variables or to evaluate the accuracy of models [WRH+00].

CHAPTER 4: PROBLEMS WITH PRACTICING STORY TEST DRIVEN DEVELOPMENT[2]



4.1 Problem Statement

As stated in Chapter 2, there are many different interpretations on the uses of Story Tests. There is some consensus that Story Tests should be used to communicate requirements. However, much is up for interpretation, such as how much testing it should contain, how it should be written, who should write it and its role in quality assurance as we have already shown in chapter 2.5. There are also different takes on what is required for story testing tools. We mention the story testing tools in this chapter, because story tests cannot exist without the tools. The tool not only influences the way customers write their story tests, but it also influences the way the test automation happens.

[2] This Chapter appeared in the following paper: Park, S., Maurer, F., A Network Analysis of Stakeholders in Tool Visioning Process for Story Test Driven Development, IEEE ICECCS 2010 15th International Conference on Engineering of Complex Computer Systems, St. Anne's College, Oxford, United Kingdom, March 22-26, 2010. The copyright release form is attached in Appendix II.

As mentioned in chapter 2.5.1 and 2.5.2, the main hurdle of adopting STDD is cost and time - how you choose your story testing tools contribute in a large part of how the story tests will be written and executed. Unlike requirements engineering in traditional software engineering, the tools in Story Test Driven Development play an important role in how the story tests are written and tested due to the automation aspect of the story test, much like unit tests in Test Driven Development. Depending on the choice of the tools for writing the story tests, the overall practice of Story Test Driven Development could be affected. Therefore, in order to discuss the uses of story tests, we cannot ignore the discussion on the roles that the story testing tool will have in the overall STDD process as well.

As stated in chapter 2, Agile software engineering does not view software development in phases. Therefore, it is inherently impossible to talk about only the requirements engineering side of Story Test Driven Development without the testing aspect as well. It is inherently impossible to talk about the test automation without some mention of the tools. It may sound strange for someone from traditional software engineering to hear that these concepts are all fused in Agile software engineering, but this is the reason why I started my dissertation with the overview of the philosophical differences. Therefore, we will discuss the artefacts, tools and the process in order to answer the objective of this chapter.

The purpose of this section is to take a sample of a wide selection of the different views on the problems that practitioners face when they try to practice Story Test Driven Development. The research is also an attempt to generate hypothesis for the rest of research. This is an inductive qualitative analysis.

In this Chapter, we present a research project which attempts to collect and analyze different experiences and evidence of Story Test Driven Development. We analyze the discussions on Story Test Driven Development from an online forum where people offered their experiences, their solutions and their view of the problems that Story Tests can solve. Analyzing these opinions and visions provides a better overview of wide array of views within the community than looking only at the published literature, because not all industry practitioners publish their experience.

4.2 Background

The Agile Alliance[3] organized workshops to envision what Story testing tools should behave like [AA07,AA08], because practitioners feel that the existing tools are inadequate for effectively facilitating STDD. Unlike Test-Driven Development that is primarily meant for developers, STDD must involve all stakeholders including customers, domain experts, developers and testers. People from different backgrounds and skills have different expectations about how one should create story tests and communicate the requirements to each other. Therefore, the issues involved in STDD are much more complex than unit testing. While such collaboration between different people has a high potential for productive and innovative outcomes, chances for misunderstanding can also be very high..

A group of Agile practitioners pursued the discussions over several face-to-face workshops [AA11]. However, coming up with a good list of requirements for the future STDD tool was a very challenging task. Therefore, a forum was created to gather

[3] The Agile Alliance is a nonprofit organization that is formed to support the advancement of Agile development principles and practices

experiences and visions of story testing tools from the community [AA11]. This is a forum where people can offer any stories, tools or visions as long as they were related to Story Test Driven Development. We analyzed the discussions in the workshops and the online discussion forum for their view on STDD. The analysis gathered about 300 features, issues, concerns and wish lists for Story Test Driven Development. There are over 350 members who are following the discussions. From this huge list of features and community members, we wanted to find out if there is a core set of concepts that are linking all of these opinions in these discussions. In addition, we analyzed the network graphs of the people with their proposed ideas to see if certain ideas have consensus in the community. The analysis could provide a guideline as to which discussion topics are popular and which may be ignored in the discussion.

The purpose of this project is to understand whether there is consensus within the STDD community. Because there are a lot of variations on the ideas, it is extremely difficult to get a big picture of what is being discussed. The motivation for our research is that the anecdotal evidence, opinions and their visions provided by the industry practitioners may provide interesting insights into Story Test Driven Development. The justification of driving innovation through online forums is the Wisdom of Crowds [W04]. The anecdotal evidence provided by expert groups is an alternative way to create insights. We can use social network analysis to extract this meaning and validate it in part by determining how consensus is reached.

This Chapter is organized as follows. In Section 4.3, research methods are presented. In section 4.4, the research design is presented. In Section 4.5, the research

results are presented. The implication of our research is presented in Section 4.6. The threats to validity of research are presented in section 4.7.

4.3 Research Methods

We used a qualitative research method for analyzing the opinions presented by the participants in the discussion forum. Strauss and Corbin state that qualitative research is a "nonmathematical process of interpretation, carried out for the purpose of discovering concepts and relationships in raw data and then organizing these into a theoretical explanatory scheme" [SC98]. Qualitative findings can be done with three kinds of data collections: (1) open-ended interviews, (2) direct observation and (3) written documents. In this research, we are using written documents for our analysis. We used a hybrid method that is inspired by grounded theory, but we also added social network analysis.

4.3.1 Grounded Theory

One of the methods used for reduction of text to code is grounded theory [GS67]. In order to build our network graph, we need to generate a set of manageable core concepts from text available on the online forum [AA11]. We used grounded theory [GS67, S87] to analyze and to reduce the discussion text to code. Grounded theory is a bottom-up research process where we start with data and see what theories/concepts arise out of that data. There are three types of coding: Open coding, Axial coding and Selective Coding. Open coding is the process of developing categories of concepts and themes emerging from data. This phase is about exploring data. Axial coding is to build connections between categories. Selective coding is to refine coded data into structured

relationships and categories. Our coded data is used along with the social network information to discover whether there is a core concept that is driving the community. A few researchers have combined grounded theory and a network analysis before [SP07, AD06]. Different disciplines use different methods for the network analysis. We decided to combine grounded theory with more rigorous network analysis based on graph theory for our purpose. Therefore, our approach is inspired by grounded theory, but we did not adopt the practice in its purest form.

4.3.2 Network Centrality

In addition to the coding, we wanted to find out how many people share similar ideas or stories. In this way, we not only get the core concepts, but how people support these concepts. We applied network analysis on our coded data based on who reported the concepts [AA07]. We used Degree centrality and Betweenness centrality to obtain the network measures. Centrality is an important concept that assigns "an order of importance on the vertices or edges of a graph by assigning real values to them". The purpose of centrality indices is to quantify an intuitive feeling that some vertices or edges on a network are more central than others [BE05]. In Centrality analysis, we are trying to discover the vertex central from vertex peripherals. In order for a graph to be analyzed for centrality, the vertices must be reachable. Reachability is defined as "the number of neighbors or the cost it takes to reach all other vertices from it" [BE05], which is also called the degree centrality. It measures how many neighbors are connected to the vertex. For a graph $G = (V, E)$ with n vertices, the degree centrality $C_D(v)$ for vertex v is:

$C_D(v) = \frac{\deg(v)}{n-1}$, where $\deg(v)$ is the number of vertices to which the vertex is linked by

an edge. The minimum possible degree is 0 and the maximum possible degree is *n*-1The definition of centrality for a graph is: Let v' be the node with the highest degree centrality in G. Let $G' = (V', E')$ be the n node connected graph that maximizes: $H = \sum_{j=1}^{|V'|} C_D(v') - C_D(v_j)$, where H is centralization of the vertex. The centralisation is the degree to which the node is central to its surrounding vertices. Then the degree centrality of the graph G is $C_D(G) = \frac{\sum_{i=1}^{|V|} C_D(v) - C_D(v_i)}{H}$. The centralisation of the graph determines the degree to which the centrality of the most central vertex exceeds all other vertices. We used degree centrality to find a group of central people who are facilitating the communication and influencing this community either as an idea leader or an idea radiator.

Clustering is a method of decomposing a set of entities into natural groups [BE05]. Cluster analysis is used when one is dealing with the types of problems where one wants to explore scattered data to discover whether a pattern of a structure exists in the data. Cluster analysis allows the researcher to discover the patterns even with the most general problem statement and measurement techniques because its main aim is to reduce the "feature dimensionality" of a search space [D00]. We used the Betweenness Centrality metric for clustering analysis [NG04]. For a graph $G = (V, E)$ with n vertices, betweenness $C_B(v)$ for vertex v is: $C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$ where σ_{st} refers to the number of shortest paths from node *s* to *t* and $\sigma_{st}(v)$ refers to the number of paths that passes through node *s* and *t* and also passes through *v.* We used Edge-Betweenness algorithm, or

also known as Girvan-Newman algorithm [NG04]. We used this algorithm because it is an algorithm that is used often in a social network analysis and serves our purpose. We used a cluster analysis to discover a set of core concepts that are important to all stakeholders.

In an open source community, people do not always engage in all discussions and they do not openly reach consensus on what is important to everyone. Often, some people are simply silent about their opinions. Therefore, simply counting the frequency of topics does not provide a good indication of consensus reached by the community. We hypothesize that we can gain much better insight on issues using a social network analysis, because stakeholders with similar backgrounds could have similar wish lists. Additionally, people's wish list may be influenced by who they interact with more often.

Most network analyses are based on the Power Law [Ba02]. The Power Law assumes that there is a strongly connected core in the network. It means there are several core concepts that connect most people. The other concepts are peripherals in the network. In our case, we suspect that the participants emphasize different issues based on what is more relevant to their current job. People with similar background and job functionality may think alike and group together more, because they tend to share similar experiences. Therefore, each of these groups may have a core idea that is different from other people.

We expect that there are multiple clusters of concepts, each with a core concept that is important to a particular group of stakeholders. People will naturally align themselves to these clusters of concepts by their job functions.

4.4 Research Design

Our research began when we participated in the first Agile Alliance Functional Testing Tool workshop [AA07, AA08]. This community keeps track of each other's progress mainly through forums [AA11] and then meet once a year. We started our data collection by going through the entries in the message board. The very first message starts on Sep 28, 2007. The data collection ended on December 2, 2008. At the time, there were a total of 536 messages.

4.4.1 Important Categories of Story Test Driven Development

First, we performed open coding on the message entries. We found that there were 226 articles that discussed important issues or concepts. The remaining articles were about announcements, workshop organizations and messages with no important discussions. The collection of these messages constituted over a thousand pages. Out of that list, we generated about 300 open codes to describe the contents. However, these 300 open codes were too granular and described too many details about the specific tool implementation features that we need to do further coding to reduce down to big concepts. Through axial and selective coding, we reduced the discussions down to 22 categories that can explain most of the contents discussed in the mailing list. The 22 categories are presented in section 4.5. And then we assigned 226 articles into 22 discussion categories. One article may be assigned to more than one category. We decided to work with 22 broader categories, because we wanted to discover a general trend in the discussion rather than specific features that people proposed.

We called the people who proposed and discussed the 22 topics as “experts” in those categories and we discovered that there are 36 “experts”. We use the term “expert” loosely. It simply means they are interested in the topic and they hold some kind of opinions on that topic. Some people appeared in more than one category, but nobody appeared in all of the categories.

4.4.2 The Research Design for Degree Centrality

Next, we were interested in finding a person who proposed the highest number of ideas that were also shared by others. The reason for this analysis is to find the person who proposed the most common ideas and analyze his/her vision for Story Test Driven Development. In other words, we want to figure out the person (or people) with the highest degree of centrality. The purpose of the degree centrality analysis is not necessarily to find the person with the most of new and innovative ideas, but the person who has the most critical social connections to help communicate the ideas across different disciplines, or to find the “deal breaker” in the community. This person would have ideas that connect with ideas that most number of people have also proposed in the community. It is also equally possible that the people who are occupying the central position are simply well versed in many disciplines and share a lot of interests with many people. We may also find whether we can use degree centrality to discover concepts that are more polarizing than others due to the division in peoples’ opinions.

The 36 “experts” are represented with vertices. These 36 “experts” are chosen, because they frequently participate in the discussion. Each time a person shares the same interest as another person, we connected two people with an undirected edge. We

performed Degree Distribution Ranking on the graph [J08]. This algorithm measures the strength of connections. It returns a local measure of the connectivity to its neighbors. The graph of Degree Distribution Ranking on our data is available in section 4.5.

4.4.3 The Research Design for Cluster Analysis

Next, we wanted to find out which categories as they were discovered in section 4.4.1 have most supporters. To obtain the core underlying concepts that are relevant to everyone in this community, we used 22 categories to form a network graph. 22 categories are used as vertices and the edges represent people's interest. Then we performed the Edge-Betweenness algorithm on the graph. The tool we used is called JUNG [J08]. This algorithm iteratively removes edges from the graph and reveals more strongly connected vertices. As we perform more iteration, we eliminated vertices with lower centrality. Semantically, it means each time we apply the next iteration of the algorithm on the graph, we eliminate less interesting concepts. The final remaining clusters of vertices are referenced and cross-referenced by most of the participants in the community either directly or indirectly through other issues. Therefore, these final clusters are the concepts are relevant and interesting to most people in the social network.

The aim of the cluster analysis is to figure out which of 22 categories are relevant to most people in the online community. We want to discover the underlying concepts that are fundamental to all of the discussions in this community. If we find that there is more than one cluster of categories, then it means the community is separated by different interests and expertise. If there is only one core cluster, then it means most participants share similar ideas and interests.

4.5 Results

In this section, we present our results. We are going to first present our codes from the text analysis and then show the results for the degree centrality and the cluster analysis.

4.5.1 Coding Results

In this section, we are going to present 22 categories that were derived from the 300 features/issues derived from the coding process. These categories summarize the major issues that were discussed by the people in the online forum. We introduce these concepts with the support of the quotations from the forums. The number at the end of the quotation is the message number in the forums.

Team Involvement

Description: Because Story Test Driven Development involves more than just developers, many contributors voiced that figuring out how to entice the rest of the stakeholders, including the developers, testers, project managers, business analysts and customers, to participate in the Story Test Driven Development is difficult.

Sample Quotations: “How to get different parts of the organization - PM, devs, testers - engaged. And how I failed in this” #2

Adoption

Description: Simply building a tool or buying a tool does not always mean all of the stakeholders will use the tools. The tool alone does not make Story Test Driven Development work, but it must accompany the practice.

Sample Quotations: "Selling such a kind of tool is like attempting to hit two balls on the same 'swing'. You have to sell the practices and sell the tool at the same time" #41

Test Maintenance

Description: Maintaining story tests in a very large project is extremely difficult. It requires additional human resources to organize these story tests, because there is no tool that can efficiently organize them automatically.

Sample Quotations: "I think teams need to understand the importance of maintainability in both their product code and their test/fixture code." #247

Economic Value

Description: In addition to writing stories and unit tests, it is sometimes hard to justify writing and maintaining story tests, which can add up to a significant cost for a large project. Without economic justification, it is very difficult to sell the idea to the management and to the team to practice Story Test Driven Development.

Sample Quotations: "There were a couple of anti-patterns that tended to tip the ROI into negative territory." #249

Regression Testing

Description: One of the many benefits of test driven development is automated regression testing. Then how do we perform regression testing using story tests and reap the same benefit as test driven development?

Sample Quotations: "You see the focusing benefit sooner - during the implementation of a story. Whereas the benefit comes after the story has been implemented." #263

Compatibility/Integration

Description: Story testing tools need to be compatible with other testing tools and easy to integrate with other testing tools.

Sample Quotations: "A shared vision of the most important next steps is… Better IDE integration? More "productized" tools ([…] RubyFIT with Fitnesse on a Mac […])" #30

Usability

Description: Story testing tools need to be able to support and communicate usability testing and its results effectively. The community used the term usability testing to refer to the automation of the user interface testing as well as the usability testing (at the mock-up prototyping stage).

Sample Quotations: "[I'm] a proponent of paper prototyping and wizard of oz testing on agile projects (code isn't the only thing that can be tested!)" #391

Communication

Description: Story testing is about improving communication of requirements between different types of stakeholders, especially when these stakeholders do not hold the same kind of technical or domain knowledge.

Sample Quotations: "Communicate and Learn seems to me most important project goals and tools on the project should support them." #169

Business vs. Technology Solutions

Description: Story testing is not a technological problem, thus trying to find a technological solution for building a better tool will not solve the problem. We need to define what can be solved by the story testing tools and what should be solved by a better business analysis

Sample Quotations: "I think it's important that acceptance tests be expressed in language, diagrams, whatever, that are independent of the technology." #131

Knowledge Representation

Description: We need a better way to represent domain knowledge in story tests.

Sample Quotations: "There are two types of knowledge: you can 'know how' to act or you can 'know that' a fact is true. Computers deal in the latter; experts deal in the former" #5

Notation/Language

Description: How do you write story tests using the notations of the domain, but also in a way that can turn into automated tests?

Sample Quotations: "I'm heavily influenced by Brian's use of dynamic language for testing." #23

Graphical Visualization

Description: Non-developers may prefer to write, view, organize and communicate better graphically. However, how do you integrate graphics into automated tests?

Sample Quotations: "We were trying to make the graphical specification more specific…and made it executable…" #58

Architecture

Description: The tests should be able to test all parts of the architecture: data, model, user interface, etc. As story testing is meant for all stakeholders, different stakeholders may want to view how the story is implemented in different layers of software architecture.

Sample Quotations: “It seems that we could run some parts of the ATs at the unit level, could be at the services level, could be at the GUI level. Each has their benefits and drawbacks.” #217

Completeness

Description: How do you know whether you have enough story tests or covered all of the testing scenarios? Does such concept have completeness apply in story testing?

Sample Quotations: “I think that implying logical completeness is asking for trouble.” #61

Distributed Tests

Description: How should story testing tools support distributed development teams and how does story testing work in distributed environment?

Sample Quotations: “To mitigate these issues a variety of strategies and tools have emerged. They primarily fall into three areas: 1. Distributed and incremental compilation and code generation grids. 2. Distributed test execution grids. 3. Selective testing tools that can dynamically construct an appropriate smoke test suite.” #466

Different Perspectives/Skills

Description: The stakeholders have different skill sets and abilities. How do you work with these different groups of people?

Sample Quotations: “I really think it's a better perspective for looking at the problem. To see it from a requirement perspective, not a test perspective.” #79

Exploratory vs. Test Automation

Description: How much of story testing should be automated and how much should be done manually? How does exploratory testing apply in story testing context?

Sample Quotations: “I do not think the skills [in TDD] are the same as traditional testing skills, nor the same as exploratory testing skills.” #222

Workflow

Description: What is the workflow for Story Test Driven Development in Agile development environment?

Sample Quotations: “We instead should focus on building tools that support a workflow. When faced with dilemma between making a tool more flexible or more simplistic, we choose a path by asking ‘which support the Agile workflow better’? #131

Abstraction

Description: Be able to capture the knowledge using the tests at different knowledge abstraction levels.

Sample Quotations: “This is all to do with the continuum between data, information, knowledge and potentially even wisdom.” #198

Terminology

Description: What is the best terminology for Story Test Driven Development? Different words for story testing mean different concepts to different people. We need to use terminologies that are clear

Sample Quotations: “On the other hand, we shouldn't eliminate the word 'test' from our vocabulary, because the 'executable examples' generally aren't sufficient to be considered a full test suite.” #196

Reporting

Description: How to report the test results to the stakeholders? What is the best way to communicate the story test results and development progress using story tests?

Sample Quotations: “Difficulty ensuring sufficient visibility and repeatability of results across the organization - Inadequate reporting, meaningless failures,…, need for archival and comparison of historical test result….” #104

Validation vs. Verification

Description: Story Tests can be used for validation and verification. Which process should STDD support more?

Sample Quotations: “System and Integration testing, however, are more concerned with the issue of 'Verification' than 'Validation'” #200

The quotations represent one of the opinions from the participants. Some are anecdotal evidences based on their experience or just a person's opinion on why he/she thinks the concept is important for Story Testing. We are not arguing for or against whether the opinions are correct.

Because this is a discussion forum with no restrictions on who participates, some topics had a very biased representation. For example, Economic Value was worded negatively only. They were suggesting the difficulty of justifying STDD to the team. No one gave a counter argument. However, some topics were given both sides of an argument. For example, Exploratory vs. Test Automation had a very heated discussion about what is test automation and how much should be automated. Some topics were proposed, but they were simply ignored by the community or misunderstood, such as Validation vs. Verification. The community quickly moved onto another topic before it received much recognition.

4.5.2 Degree Centrality Analysis

The second aspect of the research was to find the person with the most commonly shared ideas with others. The purpose of the degree centrality analysis is to discover how central a person is in the discussion if we form a network based on who proposes the similar concepts. As mentioned before, we wanted to find the people who are connected to the most people.

The vertices are the "experts" and the edges are their interests. The bigger vertices mean two things: (1) they are connected to most people due to their vast breadth of

interests; or (2) they are critical in spreading ideas because of their highly focused and specialized interest. Therefore, this graph is not measuring the persons' innovativeness of his/her ideas. As seen from the graph, this community is very tight with a lot of connections in the group. Due to the possible breach of privacy and to avoid any discomfort by the people who participated in these discussions, we withheld their names. We identify these participants through numbers and by their initials. However, because the information is available publicly, we do not feel that we had to anonymize their identity too much. The initials inside the brackets are the initials of their names.

Highest Degree Centrality: There are about a half dozen people with a highest degree of centrality. They are V1 (A.B.), V3 (B.S.), V5 (B.M.), V20 (K.J.), V26 (N.J.), V27 (N.), V28 (P.L.), V30 (P.V.) and V34 (S.T.). These are top 25% of the population. Most people who are ranked at the top only participated in the discussions a few times and expressed a narrow set of interests in the discussion forum. For example, the following is the list of their interests for each of these participants. V28 (P.L.) only appeared in Compatibility/Integration category and V34 (S.T.) only appeared in Test Automation. V27 (N.) only appeared in two of the most highly discussed topics: Compatibility/Integration and Different Perspectives/Skills. Only V5 (B.M.) is unique from this list because he was the only person who had a vast breadth of interests and contributed frequently. For example, V5 (B.M.) appeared in 18 categories out of 22 categories. As a group, the people in this top tier of degree centrality measurement are interested in Different Perspectives/Skills and Compatibility/Integration. Some of these participants have already built STDD tools previously (A.B., B.M., P.L).

Middle of the Degree Centrality: The people who are ranked in the middle of the degree centrality are the idea leaders due to their frequent participation. They are V6 (E.H.), V8 (D.V.), V11 (E.P.), V14 (G.W.), V16 (J.M.), V18 (J.S.), V19 (J.A.), V21 (K.L.), V22 (L.C.), V24 (M.L.) and V25 (M.H.), V33 (W.C.) and V35 (M.S.). This group consists of about 40% of the population. This is a very large list of people and they together have a large range of influence in the community. Most of these people are very vocal about their opinions and they participate often. However, their influence is often counter balanced with another strong idea leader. The competing interest with another contender puts them in the middle of the degree centrality. We couldn't find dominant concepts in this ranking.

Low Degree Centrality: The people who are grouped in the lower degree centrality are due to (1) their lack of participation or (2) a lot people already share the same view. The rest of the population belongs in this category. Their ideas are shared by many people in the community or there is no strong conflict with their proposal so far. Most notably, V13 (G.M.) is categorized in this category. If you look at his posting in [3, #460], he was already able to get consensus for his ideas by the community on the test automation. V2 (A.M.) appeared in this category due to his vast breadth of interest in many topics. He has 14 interested categories. Unlike V5 (B.M.), V2 (A.M.) seemed to have many overlapping ideas with rest of the community. These people are likely to be in a good position to facilitate consensus in the community, because they do not have opposing influences in the network. However, there is no dominant concept in this ranking either, because there is no one who particularly stood out and championed for an idea.

We find that degree ranking is influenced by one's number of interested topics in the discussions rather than by their job functionality. Based on the participation in these discussions, we find that the problem is getting the support from the middle-ranked participants who are in a deadlock due to their differing ideas. If the participant had a focused set of interests in the tool building discussion, they tend to rank higher. Some participants in the lower-ranked degree centrality (eg.V13) were also able to obtain a go-ahead from some people in the community although rare. It seems that consensus building is most likely to be led by tool builders at the top with a specific interest. However, the motivations and ideas tend to come from the people in the middle of the degree centrality ranking.

4.5.3 Cluster Graph Analysis

In this section, we are going to show what kind of consensus is reached in the social network analysis by analyzing which of the 22 categories were supported by most number of people. First, we counted the number of times these concepts were discussed in the forum. The frequency is available in Table 2 under the "number of messages" column. Different Perspectives/Skills appeared the most frequently with 34 appearances. This is also the topic most discussed by the people in the high ranking degree centrality. However, simply counting the number of occurrences may not provide deeper insights about the community consensus as not everyone may be participating in these discussions.

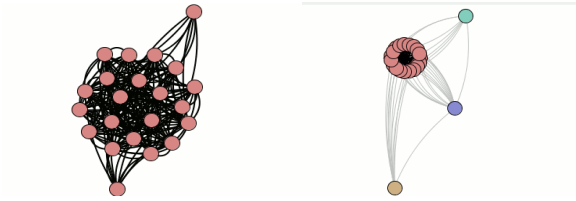


Figure 3: The graphs showing how the graph was transformed after iterations of Edge Betweenness algorithm. The left graph is the initial graph, and right graph is the final graph showing that only three categories remained

The results are found in Table 2. A lack of sub-clusters means the entire community is actually very homogeneous in terms of what they desire. Unlike our hypothesis where we assumed that people from different backgrounds will cluster around different concepts, the group shares the same visions. We did not have a threshold value, but we were interested to see what kind of clustering would emerge as we remove more edges. We determined at the end that only three categories remained: Exploratory vs Automated Testing, Communication and Business vs. Technology Solutions.

Table 2: Ranked Order of Important Concepts Using Edge-Betweenness Algorithm

Rank	Rnk by Freq	Concept	# of Msg	# of Edges Removed
1	2	Exploratory vs. Automated Testing	23	204
1	4	Communication	19	204
1	18	Business vs. Technology Solutions	3	204
2	3	Usability	22	202
3	8	Abstraction	16	199
4	18	Distributed Tests	3	197
5	13	Graph. Visual.	8	192
6	1	Diff. Persp./skills	34	188
8	5	Adoption	17	179
9	10	Workflow	12	173
10	6	Compatibility/Integration	14	165
11	12	Architecture	8	154

12	16	Valid. vs. Verification	5	141
13	9	Team Involvement	12	132
14	17	Reporting	4	119
15	6	Terminology	19	102
16	7	Economic Value	15	87
17	9	Completeness	14	72
18	15	Test Maintenance	5	57
19	8	Notation/Language	14	37
20	14	Regression Testing	6	19
21	11	Knowledge Representation	11	9

It is also interesting to note that these vertices left the core cluster one at a time as we applied subsequent iterations of the algorithm. It means there is a clear ranking of "interestingness" in the community. The lack of sub-clusters in our graph (Figure 3) shows that there are no strongly divided sub-groups of individuals who are interested in specialized topics. An extremely homogeneous group means that the group should be able to come to consensus easily, but it also means the group lacks the diversity and no focus groups exist in this community to deal with sub-topics.

4.6 Implication

The purpose of the analysis is to determine the problems that pratitioners face when they practice Story Test Driven Development, but in the process we also discovered problems with tools and processes. The analysis can provide better insights into what type of people are joining the discussions, what kind of topics are being discussed and provide some insights as to what or who may be missing in the discussions. The analysis was broken down into three sections: finding out the important categories of issues in Story Test Driven Development, degree centrality analysis of people who hold most commonly shared ideas and cluster analysis to find the most popular ideas.

4.6.1 Categories of Issues in Story Test Driven Development

The analysis categorized the discussions into 22 categories. There is a wide array of topics that people felt were relevant for Story Test Driven Development. It is interesting to note that many people applied testing concepts into Story Test Driven Development and instead of requirements engineering concepts. As one of the categories mentioned, perhaps terminologies are one of the biggest problems in Story Test Driven Development as the word, 'testing', seems to suggest to people that Story Test Driven Development should be approached with more of testing concepts.

However, in general, the categories show that people are mostly concerned about issues that arise from working with different stakeholders. For example, some of the categories that voiced such issues include communication, business vs. technology problems, different perspectives/skills, reporting, knowledge, notation/language, adoption and graphical visualization. The other categories are concerned with how to automate these story tests that may not be written in most test-friendly notations. For example, some of the categories that voiced such issues include exploratory vs. test automation, usability, workflow, compatibility/integration, completeness, test maintenance and regression testing. Finally, people voiced concern that the practice has to make sense economically. It really does not matter how good the practice is in theory if the practitioners cannot afford the resources to practice them.

4.6.2 Degree Centrality Analysis

The purpose of the degree centrality analysis is to determine if there are people with more influence in the STDD tool visioning community and then find out what their message is. The people who appeared at the top ranking of the degree centrality built their tools or expressed only a focused interest in certain aspects of the tool. The result is suggesting that perhaps the best way disseminate the practice is identify or build a story testing tool to the community, which is currently the dominant way to disseminating information on how practice Story Test Driven Development. The best example would be Fit [Fit11]. Because the practice is closely linked with the tool, the assumption is that the process will follow in the manner in which the tool can facilitate the process. These discussions emphasize the challenges of finding the right tool for writing and testing Story Tests.

The network analysis can show which ideas currently have champions. The social network analysis can show which ideas have champions through the degree centrality analysis. In our result, the top tier groups clearly suggested that there were champions for Different Perspectives/Skills and Compatibility/Integration. As shown in Table 2, the lower extreme of the ranking shows that Knowledge is lacking champions. Our findings suggest that we need to seek more information from customer's domain knowledge experts because their views are least represented in the community. It is possible that the problem is unsolved because of their lack of participation in these discussions. In addition, the champions also suggest that the biggest problem is the different skills, because not everyone, especially customers, has the necessary software engineering background to use these story testing tools. It is difficult to make customers to use story testing tools if

they do not have the same kind of skills as the developers. If the customers cannot use the tools, they will not be able to specify the story tests.

4.6.3 Cluster Analysis

We hypothesized that there will be clusters of concepts that define this community due to the diversity of stakeholders. However, our results show otherwise. There is only one core cluster with three highly ranked concepts (See Table2). Semantically, it means this is a very homogenous group and there is not much diversity in the community. Or everyone in this community believes in the same thing. What our analysis is suggesting is that despite a large diversity of individual ideas, the community tends to steer the discussion into a common theme over time based which ideas get champions.

The cluster analysis reveals interesting phenomena. First, the people in this community share similar “expertise” and interests to the point where their degree of interest can be ranked (See Table 2 for the ranking), which is certainly an unexpected result. However, we didn’t expect this community to be homogenous. The community doesn’t have many domain knowledge experts, which explains why knowledge is represented least in the product visioning discussion (See Table 2). The workshop attracts a lot of developers and testers, but it does not attract the domain experts (or the people who occupy the customer role). Therefore, we suspect that the workshop does not provide the customer’s view of story testing. It was made especially apparent from the social network analysis. It suggests that we need to seek opinions from the domain knowledge experts more.

4.7 Threats to Validity

In this section, we are going to discuss the validity of our findings in respect to internal and external validity. The study was performed on an online discussion forum organized by the Agile Alliance. Therefore, there is a risk of single group threats, which applies when the result looks at a single group. More empirical studies are needed to generalize our result with other similar discussion forums. The research also looks into one type of qualitative analysis: written documents. The written documents may not express the participant's desires very well because some people may not have participated fully in the discussions due to their busy lives. Therefore, we cannot generalize what people have written on the forum as their final words. In addition, the discussion forum tends to attract certain types of people - in this case, testers and developers. The self-selection may lead to a biased view of the software requirements.

We used our coding in a consistent manner, but other researchers may derive different codes. As Strauss and Corbin suggest, qualitative analysis is an analysis of the interpretation, but a systematic one. Therefore, we will not generalize our findings beyond what the qualitative analysis can provide to us: insights. For conclusion validity, we have shown that our network analysis has shown interesting trends as shown by the graphs and tables. However, as this was not a quantitative experiment, we present our result only as an explorative insight into the current state of STDD tool visioning process in the community. A qualitative analysis is important, because it can provide a bigger picture for phenomena. As online collaboration grows, we may be able to make better use

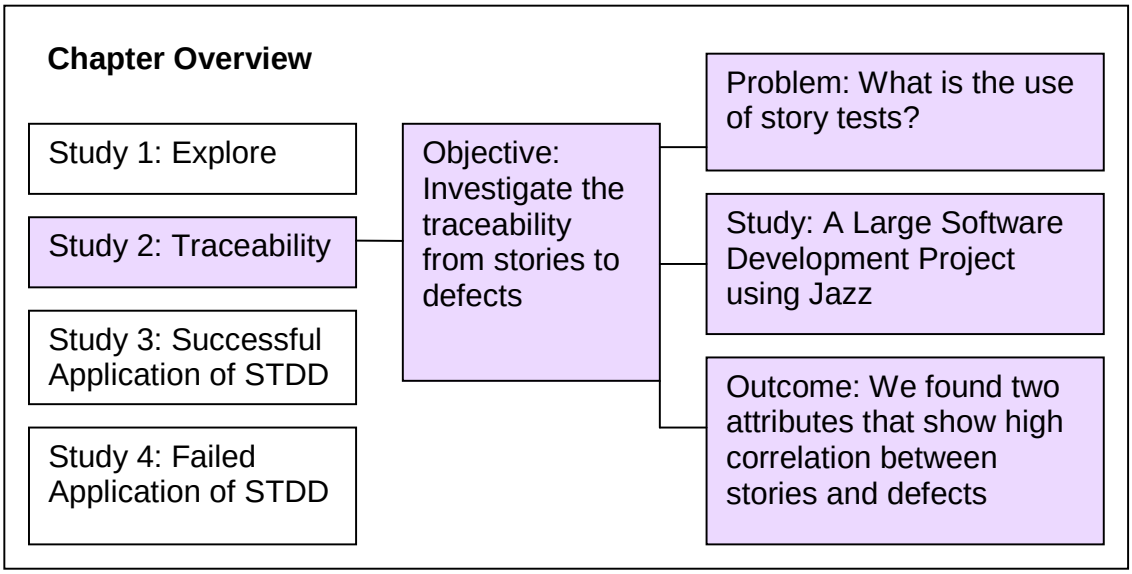
of the online community for gathering requirements and we propose that social network analysis may be one of the methods for analysis.

Basili et al. stated that drawing a conclusion from one empirical study in software engineering is very difficult, because any number of context variables could have influenced the result [BSL99]. For this reason, we cannot assume that results from this forum can be generalized into an ideal STDD tool. One criticism of empirical studies is that the result may seem obvious after the fact, but this is a misguided belief as some important facts are discovered through the evidence collected.

4.8 Summary

In this section, we presented research that was done to explore different issues related to Story Test Driven Development by analyzing feedbacks from the Agile practitioners. The research has discovered that there are largely three issues: communication due to the wide array of different stakeholders, test automation issues and economic issues. The research has also analyzed the community's social networks to see if there are popular ideas within the community. Through the cluster analysis, we discovered that three categories were very popular in the community: exploratory vs. automation issues, communication with stakeholders and business vs. technology solutions. However, some ideas were less represented in the community such as notation/language problems, regression testing and knowledge representations. We suspect that the forums had a biased representation of people in terms of their job functionalities and skill backgrounds. Therefore, we suspect that certain concepts were given much more emphasis than others.

CHAPTER 5: STORIES AND DEFECTS[4]



5.1 Problem Statement

The use of stories to communicate requirements has been widely adopted by the Agile practitioners and it is also very popular. Test driven development, where the practice of writing the tests first, then writing the code and automating the tests against the code, has been very widely been accepted within the software developer community.

Given that we are seeing thousands of new software being developed for all kinds of industries and disciplines, perhaps creating a universal story testing tools that all of these different stakeholders can use may be impossible and difficult to implement in reality. There are simply way too many variations in skills and backgrounds. In addition, Story Test Driven Development states that story tests need to be written by customers [B99]. We have to assume that most of these people who occupy the customer's role will

[4] This section appeared in the following paper: Park, S., Maurer, F., Eberlein, A., Fung, T-s, Requirements Attributes to Predict Requirements Related Defects, 20th IBM Annual International Conference Centre for Advanced Studies Research, Toronto, Canada, Nov. 7-10, 2010. The copyright release form is attached in Appendix II.

not have software development experience or training. Their backgrounds and training will be very diverse.

Another way that we may want to approach in our research is to analyze the linking aspect of story tests exclusively. One of the benefits of STDD is the traceability from stories, code and tests, which is one of the reasons why STDD is a powerful technique. The traceability from requirements to code and to the tests using fixtures (such as Fit fixtures) allows the team to identify the issues early on by identifying the relevant stakeholders, requirements, code and defects. The link between these artefacts allows the emergence of communication in regards to the design and technical issues. The link is done by writing the automated story tests using tools such as Fit[Fit11]. Therefore, we decided to approach our research by looking at what a tool with traceability capability can achieve. If we have the artefacts that represent the beginning (stories) and ending (defects), then what software development attributes would link these two factors with highest correlation. The attributes with highest correlation is what we may need to optimize with Story Test Driven Development, because story tests are the link between the two. Finding such attributes may give more insights into the relationship between stories and defects. The analysis may be able to provide insights into the hidden factors that we need to optimize using story tests.

To facilitate this research, we analyzed data from a development team that used stories, test driven development and an organizational tool that has the capability to trace from stories, code to defects, but they did not use Story Test Driven Development. In our case study, we used the Jazz development tool as our study object [Ja11]. The Jazz functionalities have the potential to address the problems that people mentioned in

Chapter 4 either directly or indirectly, such as Abstraction (organize the project based on teams and components), Communication (dashboard), Exploratory vs. Automation Testing (unit testing and creating work items for each test), Compatibility/Integration (though Jazz APIs), Architecture (component view), Validation vs. Verification (dashboard), Team Involvement, Reporting, Economic Value (easy to buy the Jazz platform, intuitive to use and it can be readily used out-of-the-box), Distributed Tests (Jazz is a server based technology), Graphical Visualization (Jazz comes with a user interface), Completeness (integration with the unit tests), Test Maintenance and Regression Testing. Validation vs. Verification or Exploratory vs. Automated Testing would be embedded in the workflow process rather than functionality of the tool. However, it is something that can still be addressed within this type of work environment.

Jazz offered a lot of attributes that can be data mined to get better understanding of the development progress. We want to find out if stories can be linked all the way to the defects and vice versa. If so, then we want to find out whether a tool such as Jazz can tell us which attributes are most relevant for the number of defects at the end. Our research will help shed more insight into what traceability alone can achieve and what Story Test Driven Development should achieve in addition to traceability in order to stand as a separate but critical technique that no other techniques can replace.

The organization of the Chapter is as follows. Section 5.2 motivates our research with the background information. Section 5.3 describes the development project, the data of which we analyzed. Section 5.4 describes the research design. Section 5.5 describes the results. In Section 5.6, we discuss our findings. Section 5.7 describes the threats to the validity of our research. Section 5.8 summarizes the findings from this section.

5.2 Background

The aim of the research project is to data mine a structure on the relationships among requirements, people and software defects in large software development project that used stories, test driven development and a project dashboard.

The discipline of statistics and data mining are both concerned with discovering structures in data [H99]. A large body of data may contain some valuable information which may provide more interesting or useful insights into a phenomenon under study. Statistics is generally concerned with how to make statements about a population by examining a sample of the population. On the other hand, data mining is concerned with an entire population. In such situations, statistical model building is used to find significance of the model fit rather than the probabilistic statement about the generalization ability from a sample [H99]. Data mining deals with searching for variables that may have good predictive abilities and try to find potential explanatory variables. In data mining, we are more interested in the *exploratory* aspect of discovering potentially interesting relationships between many variables. In contrast, statistics is a *confirmatory* analysis that builds a model derived using theoretically selected variables applied on a sample [H99].

In this Chapter, we report the results of data mining a software repository of a team that uses stories for communicating requirements. We data mined their development repository covering over a year of project lifetime to find out if any interesting structures or patterns exist on the relationships among requirements, people and defects.

We aimed this part of research at defect prediction and decided to work from defects to requirements backward to find out if defects can be traced to specific stories. Once we have the defect prediction model, we can determine which attributes have the highest correlation when they are traced from defects to the relevant stories.

Our aim of producing the defect prediction model is different from traditional defect prediction research that is based on code. Defect prediction is a research area with a long tradition that aims to find metrics that are available in the early phase of software development and that are good defect predictors [MPS08]. However, defect prediction research tends to evaluate defects from the coding stage onwards using attributes that are available at the coding stage, such as code churn, lines of code or the number of file changes [NB05]. Our purpose is to find the relevant attributes at the requirements stage that are related to stories in order to find out whether such attributes may also have relevant in STDD environment.

Based on the Jazz system's change history and the people who work on the project, we hypothesized that easily attainable requirements-related attributes could exist in our data that have high correlation with defects count. Our reasoning is that stakeholders may hold *tacit knowledge* about a project's health, which may manifest itself in some *human-based attributes* that can be measured at the time of the requirements specification. Therefore, our hypothesis is that there are measurable story attributes that can provide reasonable defect prediction. We are interested in what these attributes are in order to investigate whether and how they are related to Story Test Driven Development in our subsequent research.

In the following sections, we present two approaches to this particular research.

5.2.1 Defect Prediction

A defect is a common terminology for a *fault* in a program [Ma08]. However, in our study, we simply identified defects to mean the defect workitem. Any work items that were labelled as defect work item by the team is considered a defect.

Defect prediction is a research area that aims to answer the following questions [ME98]: 1) Find metrics that are available in the early phase of software development that are good defect predictors; 2) Develop models that can be used for defect prediction; 3) Evaluate the accuracy of the model; 4) Calculate the cost of utilizing the model in a software organization. Defect prediction requires various kinds of knowledge repositories that can be easily mined for obtaining the status of the project at a given point in time. People have used fault databases, code repositories and feature request databases for their defect prediction analysis based on code [ME98]. For example, some of the work done in defect prediction includes [ME98, NB05, K93, OWB04, KAGNM96, ZN04]. Currently, the estimation for these code-based predictors for defects has reported success rate of up to 70% to 89% [NB05, ME98]. Nagappan also examined the rate of *code churn* on Windows Server 2003 code and determined that it can predict with 89% success rate where defects will be found.

Unlike these research projects, our aim is to find out whether there are attributes present in the requirements specification, particularly stories, that can be used to trace to the defects using stories and their relevant attributes. Our aim is not about deriving a defect prediction model that can predict better than code-based models. We are simply inspired by their research models and applied their methods to our research. As far as we

know, defect prediction using requirements specification attributes available in stories has not yet been attempted.

5.2.2 Network Analysis

The motivation behind network analysis is to understand the structure and evolution of the relationship between entities. Part 1 of our research suggested that there is a strong correlation between the number of stakeholders and the defects at the end, which we will explain further in the subsequent Chapters. Therefore, for part 2 of our research, we wanted to understand better the relationship between these people based on the trail of artefacts that they left behind and their team organizations. Our main interest in employing network analysis is to analyze the chronology of how the stories eventually created defect work items and then discover which network attributes had the closest network distance linking these two artefacts.

Many natural networks have a few nodes that have many more connections than the average node has. Therefore, most real world networks emerge using the Power Law [Ba02]. Network analysis can also be performed on software engineering artifacts. Zimmermann et al. performed a network analysis on binaries of a single project [HH04]. Hassan et al. looked into research trends by performing network analysis on the reverse engineering community [JSG+06]. Jacovi et al. identified sub-communities within the CSCW research community using network analysis on published research papers in the past CSCW conferences [S66]. The network of how people are related to each other based on their team organizations and work items may provide better insights into whether social networks contribute to the defects in some predictable ways.

One of the important concepts in network analysis is the difference between *ego network* and *global network.* An ego network is concerned with its immediate neighbours. Each node in the network has an ego network. A node is often called “ego” in network analysis [F07]. The global network looks at the entire nodes. We want to find out which type of network is more relevant for the relationship between stories and defects.

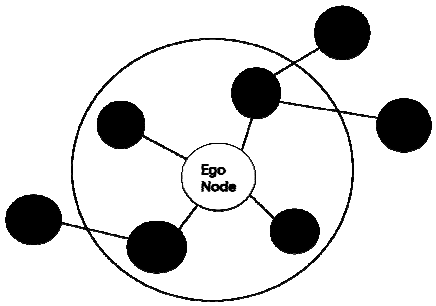


Figure 4: The nodes inside a large circle are the ego network for the node located in the middle labelled as ego node. A global network refers to all the nodes in the picture.

5.3 Case Study

We used the data large software development project from a company that we will only identify as Company A for our analysis [CHRP03, CHS+03]. We obtained a large repository of data that contained information on how Company A produced their product. The repository has many variables that can be explored, which provides an interesting case study for data mining. As mentioned, data mining is an exploratory analysis. Our aim is to find out whether any interesting structures or patterns exist in the given data that may provide insights into the relationship between requirements (stories) and defects and whether such information is embedded in the repository produced by the Jazz team.

In the rest of the section, we will explain the Jazz tool Jazz is a development environment produced by IBM to support collaboration on software development. The Jazz system includes an integrated programming environment as well as communication and project management tools [CHRP03]. Company A had a substantial repository of development data that was created during their development project using Jazz. We data mined their repository for our study. The data we extracted includes data from December 8, 2006 to June 26, 2008. 151 contributors (user accounts) exist in our data, but only 93 unique users were relevant to our study because others did not participate in the project during our time frame in terms of work items that can be traced from stories. The active users are anyone who were part of the development team and held a user account to access the repository. The teams are distributed over 16 different sites, including the United States, Canada and Europe. Seven of these sites were active in the development and testing. The development project had 90 components.

Company A used the “Eclipse Way” methodology for their development [CHRP03, CHS+03].. Each iteration consists of six weeks. A project management committee sets up a goal for each iteration and breaks down the goal into features, called work items. A *work item* represents an assignable and traceable task that can be categorized into different subtypes, such as defect work items, story work items, enhancement work items or retrospective work items. Each work item was then assigned to a development team. In addition, each work item had a specific owner who tracked the work item from the beginning to the end. However, many people could contribute to the work item, such as contributing to its implementation, joining discussions and subscribing to the work item to keep an eye on its progress. Developers who are only

“subscribed” to a work item do not contribute to its implementation but they keep track of its development because it might be relevant to their own work. Each work item contains information about its time of creation, its time of completion and the person assigned to deal with it. The team coordinates the implementation effort by commenting on work items.

One of these work item types is Story. This is the work item that specifies a goal (or a requirement). Another type of work item is Enhancement. When we mention a *story*, we mean both Story and Enhancement work items. If a defect is found, a work item of type Defect is created. Here is an example story.

Provide an integration option for the Visual Studio client:

Our current option for writing an SCM client is to use a combination of server REST and command line tools. It turns out that the command line (CLI) is both too slow and far from feature complete. In addition, calling the CLI and parsing stdout isn't an option for providing a rich integration into another IDE. This story is about enhancing the client side integration to allow:

- a rich and feature complete visual studio client or any other client written in Java or another language.

- an integration which is as fast as the current RTC UI client.

- an integration architecture which ensures that feature X added in the RTC UI can easily be leveraged in client Y.

- a CLI with feature parity with the RTC UI for improved command line usage and scripting.

In the case of a defect, the team goes through a short discussion phase to make sure the work item is indeed a defect before it is assigned to a team member who will fix the defect. If the work item is vague, the team members can ask questions for clarification. The Jazz system allows work items to be linked to each other if they are conceptually related. For instance, if a defect can be traced to a requirement, these two work items can be linked. We use these links to determine if a defect is a requirements-related defect. i.e., for each individual requirement (work item), we measure the number of linked defects. The developers may not have included all links, but our data size is large enough to have significant results despite possible missing links.

In our research with the Jazz development data, defect refers to any work items that are labeled as defects in the repository. Requirements-related defects are defect work items that can be traced to a story work item. We noticed that 94% of the defects can be traced to one or more of the story work items. The rest of the defects appeared without any relationship to stories. In the Jazz repository, a story work item can be – and often is – linked to several defect work items. The process of linking can lead to very complex links of defect fixes. The Jazz system maintains these links as a part of the tool's functionality. What we are interested in is whether there are some overarching structures or patterns that arise from these individual links between work items.

Developers report the relationship between work items because they think that knowledge about the existence of other work items may help solve the problem. In other words, the links between story work items and defect work items are based on human understanding of the problem, not an automatic association generated by the code. In this

sense, the linking between defects and story work items is different from linking of the change sets to the bug reports or linking of bug reports to failed tests, which are automatically generated by Jazz through code submissions. Because the linking of stories to defect work items represents a human understanding of the problem, there is no absolutely right or wrong way to interpret what the linking may mean other than that the developers who were assigned to the task thought that they were relevant and important. When individual developers make these links to help solve one's own problem, it may eventually emerge into a network of linked work items that may together have greater meaning. In addition, there is no valid instrument to check whether these links are right or wrong. These networks are different from networks that you would generate from code dependencies or other code-based metrics, because the links are inherently based on qualitative reasoning that came from each developer.

Once the work items are generated and assigned, the code must be submitted and managed through Jazz. To coordinate the implementation integration, each team commits their source code to their Stream. Streams allow each team to keep different versions of components in order to make their implementation independent of other teams' changes. A continuous integration process takes place at a team level. Each team makes changes to the code using Change Sets. Once the team has a stable version of source code within their stream, they commit their work to the Project Integration Stream. A build can be produced from each of these streams according to set periods of time. We combined all of these workspaces and obtained the latest state of the code as well as the code history for our analysis. The entire data, including work items, any file attachments, code base, all of their history, is about 21.3 GB. The code base consists of 2.34 GB of data. Our

extracted data includes data from December 8, 2006 to June 26, 2008 with a total of 2,860 story and enhancement work items and their 215,099 related defect work items.

5.4 Research Design

We modeled our research similar to [NB05, ZN08]. However, we looked specifically at non-code attributes and considered the types of information that are available at the requirements level. We used the Jazz web interface as well as Jazz Team Concert (which are tools within Jazz) to extract data relevant for our analysis. Then we built a script that calculates the metrics used in our investigation.

We present all the variables that we explored whether they had correlations or not. Data mining is about exploration; thus, negative correlations are as interesting as positive correlations. As mentioned previously, data mining is about discovering relationships. We categorized our variables for the purpose of obtaining explanatory power, but the results were originally obtained from an exploratory process.

For the purpose of presentation, we categorized the attributes into point and aggregate variables. The point variables are attributes whose values can be obtained from a single story. For example, a time estimation to implement a story is available from a single story. The team may decide to change the time estimate at a later time if new information becomes available but the estimation information is still available from one story.

On the contrary, aggregate variables are values that are accumulated across multiple work items. Therefore, it is not possible to obtain these values using only one value from one story. For example, the number of related stories may change over time as

new requirements are added to the project. Therefore, a story with only one related story may have two or more related stories at a later point in time as additional requirements are added to the project.

In our study, the dependent variable is the number of defects and the independent variables are the point/aggregate variables that we are going to present below. We are trying to map one-to-one relationships between each of these variables to the number of defects. In other words, we are measuring whether these independent variables have an influence over our dependent variable, the number of defects.

5.4.1 Point Variables

For the point variables, we have the following attributes for each story:

1) Time Estimates: The time estimates are developers' estimation of how long a story will take him/her to implement. Not all of the stories had time estimate information available. Our assumption for data mining the time estimates is that some tacit knowledge about the difficulty of the implementation may be reflected in the time estimates; some structures or patterns may arise from the estimation trends. For example, stories that are expected to take longer to implement may be more difficult (e.g. more complex or simply more comprehensive), thus could be prone to more defects. Since we do not have code complexity information at the requirements stage (as we stated that we designed our research to only consider information available prior to coding), time estimates may provide an alternate way of predicting the developers' projection of the possible code complexity.

2) Priority: Priority is measured as “Unassigned”, “Low”, “Medium” or “High”. This measurement is the stakeholders’ view of when the story should be implemented in comparison to other stories. A story that is assigned a high priority should be implemented before a story that is assigned a low priority.

3) Ownership: Each work item usually has an owner who makes sure that the work item is finished. This is usually the person who finally signs off the work item as resolved, although not always. We can interpret the correlation to mean that someone who owns many work items may have better knowledge about the stories’ health because he/she has an understanding of how different work items are integrated together. Or he/she may have a better idea where defects come from, which may not be so obvious in a large project. On the contrary, a person may get overwhelmed by many work items and then make mistakes. Either a positive correlation or a negative correlation would confirm that prediction can be made based on this variable. No correlation means the ownership is a poor predictor for the possible number of defects.

5.4.2 Aggregate Variables

1) Number of Indirect Stakeholders for the Story: We define a stakeholder as any user who had an account in the project repository. This includes developers, user interface designers, requirements analysts, testers, project managers, etc. We define indirect stakeholders as people who report defects but have not been involved in the initial definition of the story. For example, a story work item has owner and people who contribute to the discussions. If the related defect work item is reported by someone who did not appear in the story discussion, this person is identified as the indirect stakeholder.

If there is a positive correlation, it suggests that defects arise due to not recognizing the true extent of the indirect stakeholders. It could also suggest that people did not realize how the introduction of the new story will influence someone else's work. A negative correlation would suggest that having more indirect stakeholders actually leads to less number of defects. If such a trend does appear, we may have to investigate more as to why. No correlation would mean that this variable is not a good indicator for obtaining defect predictions.

2) Number of Related Stories Based On Shared Defects: We identified those defect work items that are linked to two or more story work items. We interpreted these defects to mean that there were unexpected interactions between requirements. If there is a positive correlation, there is strong support that defects arise due to unexpected interactions between requirements or a larger network of interactions between work items. If defect fixes require knowledge about other story work items, the person who implements the fixes needs to consult with other team members. The need to be aware of many work items could mean that there is more potential to change the behavior of requirements that someone else wrote in an unintended way. If there is no correlation, it suggests that story interactions do not provide a predictable trend that can be used for defect prediction.

3) Story Creation Time in Relation to Defects: We decided to test whether introducing a story at a later time (after some iterative code implementations) leads to more defects. We measured the time when a story was introduced to the project and measured the subsequent number of related defects. While introducing new stories later

in the development stage does not directly measure requirements change, it does represent a lack of such requirements information before they were introduced. Since some implementation had already happened before these new stories are introduced to the team, the developers may have designed code without the knowledge that such requirements may be coming up in the future. Three possible scenarios could exist. First, stories that were introduced earlier in the project could end up getting more related defects as time progresses, because new requirements may undo the original implementation. Second, the new stories could have a higher number of defects, because the new implementation has conflicts with the older implementation. Or there may be no correlation. Again, a correlation can provide some clues as to whether the "Story creation time in relation to defects" provides trends for the purpose of predicting defects.

5.4.4 Null Hypothesis

The null hypothesis states that there is no correlation between any of the six attributes suggested above and requirements-related defects. Literature suggests that a *p*-value below 0.05 is considered to have high statistical significance [FPP98]. If the statistical significance is below 5% [W71], we are going to suggest that the alternative hypothesis, which is that there is a correlation between the selected attribute and the occurrence of requirements-related defects, is supported. Based on our data, we cannot absolutely prove the relationship between the attributes and the occurrence of requirements-related defects, but it may suggest that there may be a strong relationship. We are looking at the attributes individually without considering possible interactions

between attributes. Therefore, we are going to perform the analysis on each attribute separately.

5.4.5 Network Analysis

For Part II of our analyses, we look at the network patterns in our data based on the attributes that show the highest correlations. These associations between work items and people are drawn up into a large network graph. The purpose of the network analysis is to provide explanatory patterns as to how attributes are related to each other. We measured the following attributes for ego networks:

Size: The size is the number of nodes in the ego network. It includes nodes that are one step away from the node, $n_{i.}$

Two-step reach: The two-step reach measures the percentage of nodes that can be reached in two directed steps from the node.

Brokerage: The brokerage is the number of times the node appears in other nodes' connection paths. The brokerage value would be high for a node that is connected to many nodes, because it can play the role of a broker in connecting two unconnected pairs. The measurement is obtained for each ego node.

Effective Size: The effective size is measured by the number of its neighbours minus the average number of directed ties between these nodes. Let's suppose there are three nodes, n_1, n_2, n_3, and n_2 and n_3 have a directed connection and n_1 has a directed connection to n_2. The effective size for n_1 is 2-1=1.

We measured the following attributes for the global networks.

Degree Centrality: The degree centrality measures the number of dependencies for each stakeholder. For ego networks, we measured *In-Degree*, *Out-Degree* and *InOut-Degree* of a node. *In-Degree* measures the number of incoming connections to the node. *Out-Degree* measures the number of outgoing connections to the other nodes. The *InOut-Degree* is the sum of *In-Degree* and *Out-Degree*.

Betweenness Centrality: The betweenness centrality measures how many times the node appears in the other nodes' shortest paths calculations. First, we need to calculate the probability index of communication paths between two nodes. If the network offers more than one shortest paths between two nodes, n_j and n_k, then all of them have the same probability to be chosen. Suppose one of these shortest paths contain the node, n_i and let $g_{jk}(n_i)$ be the number of shortest paths linking n_j and n_k., then the probability that n_i is beween n_j and n_k is $g_{jk}(c_i)/g_{jk}$. Then the betweenness centrality is measured using the following formula:

$$C_B(c_i) = \frac{\sum_{j<k} g_{jk}(n_i)/g_{jk}}{(g-1)(g-2)}$$, where

C_B is the degree centrality and g is the number of nodes in the network.

Finally, we are interested in the team assignment of the stakeholders. Instead of categorizing developers individually, we put them into teams. There are 90 project components. Each component is assigned to a team. Each team has its own stream. A stream is a workspace with a separate branch in the source repository. Each team commits their code into their stream only. We wanted to see if dependent defects are found by the members inside the team or members outside of the team.

Percentage of People Outside of the Team: In the ego network, we want to find out how many of these connections are with people outside of their team. If this value is high, it denotes that requirements-related defects are mostly found when there is an interaction with outside teams.

Associated Team Areas: A person is assigned to many team areas or none at all depending on their job description. We want to know if a person assigned to many teams and overseeing many projects could detect more requirements-related defects.

5.5 Result

In this section, we describe the results of the case study performed on a large development project in Company A that used Jazz. Section 5.5.1 presents the correlation analysis between the requirements attributes and the code attributes. Section 5.5.2 presents the regression analysis and section 5.5.3 presents the data splitting in order to measure the ability to predict system defect density.

5.5.1 Correlation Analysis

We used Pearson correlation coefficient [FPP98] to verify the correlation between the specified attributes and defect occurrences. Pearson correlation is preferred when we are working with raw data. The closer a correlation value is to -1 or +1, the higher the correlation between the two attributes: +1 means they are perfectly positive correlated and -1 means they are perfectly negative correlated. A value of 0 indicates that the two measures are uncorrelated.

The Pearson correlation values are shown in Table 3. We based our threshold on Colton's rule of thumb for interpreting the size of correlations, which is follows [C74]:

Correlations from 0 to 0.25(or -0.25) indicate little or no relationship; those from .025 to .50 (or -0.25 to -0.5) indicate a fair degree of relationship; those from 0.50 to 0.75 (or -0.50 to -0.75) a moderate to good relationship; and those greater than 0.75 (or -0.75) a very good to excellent relationship.

While the correlation coefficient measures the *strength* of the relationship, the significance measures the probability of an event occurring by chance only. The significance is measured using a probability level denoted as *p*. A smaller *p* means that the result is unlikely to be caused by pure chance. As defined in the research design section, a *p* value that is smaller than 5% is considered significant for our research and we will reject the null hypothesis [C74].

The result is presented in Table 3. To summarize, we observe that there is a strong correlation relationship between the number of defects and the

- Number of Indirect stakeholders
- Number of Related Stories

The significance value for Story Creation Time in Relation to Defects is higher than our threshold of 0.05; therefore, we cannot make any general conclusion about this variable and it is eliminated from the candidate variable. However, the other variables provide high significance values.

In terms of Time Estimates, Priority and Ownership, our data shows that there is clearly no relationship between these variables and the defects count. They all show high statistical significance to support our observation. See Table 3.

Based on our result, the two variables in the aggregate variables category, the Number of Indirect Stakeholders and the Number of Related Stories, both show high correlations with the number of defects. The point variables all show no correlations with the number of defects.

Table 3: Correlation coefficient between the specified story attributes and the number of defects

Attributes	Pearson coefficient (*r*)	R^2	Significance (*p*)	Mean	Variance	Std. Dev.	Std. Err.
Time Estimates	-0.0222	0.001	<0.01*	1119.83	1,463,879.2	1209.91	110.91
Priority	0.0602	0.004	<0.01*	2.04	0.07	0.27	<0.01
Ownership	-0.0316	0.001	0.04*	772.86	1,302,499.28	1141.27	21.31
Number of Indirect Stakeholders	0.9048	0.819	<0.01*	5.36	37.56	6.13	0.11
Number of Related Stories	0.7591	0.576	<0.01*	17.01	1,417.89	37.6	0.70
Story Creation Time in Relation to Defects	0.0144	0.001	0.22	332.41	22,465.18	149.88	2.8

*Significant at alpha=0.05 level

Table 4: Regression Analysis

Attributes	Regression Model	Standard Error of Estimate	MMRE[5]	PRED[6] (0.3)
Number of Indirect Stakeholders	$y=\frac{x^{1.56}}{3.55}$	0.30	0.27	0.76
Number of Related Stories	$y=(0.009x+1.244)^6$	0.37	0.20	0.87

[5] MMRE: Mean Magnitude of Relative Error

[6] PRED(*p)* : Prediction at level p where p is a percentage. It refers to the number of cases in which the estimates are within the p limit of the actual values, divided by the total number of case.

5.5.2 Regression Analysis

The purpose of a regression analysis is to develop an equation of a line that best fits most of the data points. The Standard Error of Estimate is calculated to check for the discrepancy between the data and the regression model [W71]. This is the distance between the actual data points and the regression line.

At this point, we narrowed down our analysis to the two variables that have shown high correlation coefficients: the Number of Indirect Stakeholders and the Number of Related Stories. Therefore, we performed the regression analysis for the two attributes that show statistical significance and strong correlation.

Our analysis shows that a power regression and a polynomial regression fit our data best as seen in Table 4. The number of indirect stakeholders has a good regression model with a relatively small Standard Error of Estimate. The correlation analysis and the regression analysis both confirm that there are indeed positive trends in the relationship between these two variables and the number of defects that are beyond a random occurrence of events. MMRE of <0.25 and PRED (0.3) of >0.75 are considered to be highly acceptable model of accuracy [PK08]. The MMRE and PRED(0.3) in our analyses are both in the range of highly acceptable numbers, which suggests that our regression model can fit our data with good accuracy.

5.5.3 Data Splitting

Based on our analyses, the variables that show consistently high correlation with the defects are the Number of Indirect Stakeholders and the Number of Related Stories. Therefore, we used the data splitting technique on the Number of Indirect Stakeholders

and the Number of Related Stories. Data splitting [NB05] is a technique to independently assess the ability to predict from a population sample. We randomly select two thirds of the stories (1906 stories) from a population to build the prediction model and then use the remaining one third (954 stories) to verify the prediction accuracy. Then we find out whether our variables still hold the prediction ability even with different training and evaluation values.

Using the regression equation, we estimate the defect density for the remaining third of the stories. Then we compare the estimated values with the actual values for the remaining one third of the stories that were used for the evaluation. We ran the correlation analysis between the estimated and actual values. A high positive correlation coefficient means there is a positive relationship in the attributes being measured and the estimated defect density.

All trials show consistent positive correlation and statistical significance as shown in Table 5 and 6. The magnitude of the correlation provides the sensitivity of the predictions. A higher correlation means the prediction has a higher sensitivity. The result shows that the Number of Indirect Stakeholders is a very good predictor of defect density and the Number of Related Stories is a moderate to good predictor.

Table 5: Data Splitting Regression and Correlation Analysis for Number of Indirect Stakeholders

Trial #	**R^2**	**Significance (*p*)**
Random 1	0.8248	<0.01*
Random 2	0.8102	<0.01*
Random 3	0.8199	<0.01*

*Significant at alpha[7]=0.05 level

Table 6: Data Splitting Regression and Correlation Analysis for Number of Related Stories

Trial #	**R^2**	**Significance (*p*)**
Random 1	0.5432	<0.01*
Random 2	0.5628	<0.01*
Random 3	0.6294	<0.01*

*Significant at alpha=0.05 level

5.5.4 Networks of People and Stories

The result so far suggests that the two variables, the number of indirect stakeholders and the number of related stories, can predict the number of defects very well. From a statistical perspective, it suggests that these two variables are good predictors of the number of defects. The next question is how these two variables relate to the number of defects and provide some characteristics about their relationships. To

[7] Coefficient alpha is the probability that you will wrongly reject the null hypothesis. It is also referred to as a false positive.

evaluate the nature of their relationships between the stakeholders and stories, we analyzed how each person (stakeholder) is linked to stories. Two people are linked on a network if they both share some work for the same story. Table 7 shows the statistical analysis of how these values relate in terms of the network measures.

As shown in Table 7, size, two-step reach, brokerage and effective size are showing high correlation. Table 7 also shows that the betweenness measure shows very high correlation, but the degree centrality measure does not. What this means is that the person who can explain the most number of related stories using the smallest number of related stories (in other words, shortest path between stories networks) is the most important person, not the person who is linked to the most number of stories. Finally, the Percentage of People Outside of the Team and Associated Team Area show moderately positive correlations. It means that there are some reasonable trends that defects are discovered by people outside of the immediate core team and more likely to be detected by people who are working on multiple teams.

Table 7: Correlation Analysis on the Network Measures for Stakeholders and Related Stories

Measures	**Pearson Coefficient (r)**	**Significance (*p*)**
Size	0.9182	<0.01
Two-Step Reach	0.9367	<0.01
Brokerage	0.9096	<0.01
In Degree Cent.	0.4356	<0.01
Out Degree Cent.	0.26625	<0.01
InOut Degree C.	0.2617	<0.01
Betweenness	0.5130	<0.01
Effective Size	0.9182	<0.01
%Outside Team	0.6775	<0.01
Associated Team	0.5475	<0.01

5.5 Discussion

In this section, we discuss the results of our analyses. We have shown through our analyses that the Number of Indirect Stakeholders and the Number of Related Stories can predict the *trends* in the number of defects. In addition, we have shown that stakeholders and stories are related in terms of size, two-step reach, brokerage, effective size and betweenness centrality. In addition, we have also discovered that people outside of the core team may find more defects (than the original team members) as well as people who participate in the development of more than one component.

The other attributes, such as time estimation, priority and ownership, did not show that they were good predictors for the number of defects. The “Story Creation Time in

Relation to Defects" did not meet the significance threshold; thus, this value is inconclusive. However, what is important from our findings is that trends for the number of defects can be predicted from requirements attributes. In this section, we are going to discuss the implication of our findings in terms of what this could mean for defect prevention.

5.5.6 Indirect Stakeholders and Related Stories

The two attributes that show very high correlations with the number of defects are the Number of Indirect Stakeholders and the Number of Related Stories. Both of these numbers are quantitative; therefore, they can be measured at any given point in time. However, both of these are aggregate variables, which mean these values cannot be measured alone or only at one point in time. Rather, they grow and change as more stories and people are added to the project.

The findings can be interpreted in many ways. Our result suggests that the interaction of people really matter in explaining the number of defects. There is no doubt that people are important in requirements engineering. Cheng and Atlee states that "successful RE involves understanding the needs of users, customers and other stakeholders; understanding the context in which the to-be-developed software will be used; negotiating and documenting stakeholders' requirements and validating that the documented requirements match the negotiated requirements" [MR96]. This definition emphasizes that there are a lot of human social activities involved in RE, such as identifying the needs and negotiating for agreements.

Other literature suggests the importance of stakeholders by proposing various methods which are designed to reduce conflict between ideas of different stakeholders. They include group elicitation techniques, such as brainstorming [D92], prototyping when dealing with a great deal of uncertainty [SRP07], cognitive techniques, such as think aloud [SRP07], card sorting [S01, S04] and contextual techniques, such as ethnographic studies [SRP07]. The study confirms existing knowledge that there are relationships between stakeholders, stories and requirements-related defects.

Our analysis suggests that we need more studies in identifying and understanding the stakeholders on how they communicate and how they are involved in the teams.

5.5.7 Network Analysis

The second part of the analyses is to find out how these two variables, the Number of Indirect Stakeholders and the Number of Related Stories relate to each other. To understand the characteristics of their relationship, we measured 10 network attributes. As shown in Table 7, four attributes show very high positive correlations and three attributes show moderately positive correlations. Other attributes do not show any correlation.

First, the ego network values all show very high correlation values. This suggests that one's own knowledge about the people who are working on the related stories is very important in predicting the number of defects. If a person is working on a story that associates a lot of people, then it is likely that this story is also related to many defects. It shows the need for developers to find people who are working on similar or related stories in the team as early in the development as possible.

A person may be assigned to multiple team areas depending on their functional specializations or their breadth of knowledge. Our result suggests that most of the defects are discovered by people who did not belong to the same team as the person who created the story. A moderately positive trend shows that defects are sometimes discovered by people outside of the team. Finally, we measured the total number of teams represented per stories. Our result moderately supports the proposal that a component with people from many different teams do end up with more defects.

5.5.8 Predictability

The result does confirm our hypothesis. There are requirements attributes that have strong correlations with the number of defects. Even at the requirements stage, the number of indirect stakeholders plays a crucial role in the defects count. Making sure that everyone knows how changes will impact their part of the work may be important.

5.6 Threats to Validity

In this section, we discuss the validity of our findings with respect to internal and external validity.

For conclusion validity, we have shown that our result has very high statistical correlations. We are not arguing for multivariate statistical significance of our result. Each attribute is measured independently from other attributes. We are only working with one project, so there is no risk of random heterogeneity of subjects. It also means that the statements are valid for the project under study, but we cannot generalize our result to all projects without further studies.

We assumed that the team members in the project made their best effort to link the defects to the requirements and documented all of their effort. However, human error may have led to excluding some defects because the developers may not have linked the defects to the requirements. However, we believe that the data is sufficiently large enough to warrant statistical analysis of our result. We also assume that everyone in the team consistently used the repository to communicate and record their development progress.

For construct validity, we made considerable effort to discuss the limitations of our attributes and any assumptions we made to obtain the measurement. We measured multiple attributes in both point and aggregate categories. There is no participants' bias toward the research by the people who participated in the project, because the data represents their normal development progress, rather than a response to a research study. The selection of measurements was based on a literature survey as well as on what was available in the data set.

Depending on when a feature was developed, some data might not be available. Before the concept of Story work item was introduced, Jazz was already keeping track of some defects. We ignored these defects from our analysis if they did not relate back to one of the Stories or Enhancements. However, 94% of the defects are accounted for through the relationship between Stories and Enhancements. In terms of the population selection, our data had 93 unique contributors, which is large enough to account for any natural variation in human performance.

One of the threats to internal validity is the interpretation of the causal influence. Based on our analysis, we can state that there is a strong correlation between the Number

of Indirect Stakeholders and the Number of Related Stories to the number of defects, but we cannot suggest that these attributes can cause defects. In addition, defect prediction using knowledge, code, or defect repository does not show any social dynamics that may exist in the team under study. There may be other social factors that are invisible from the knowledge repository that may account for the numbers.

For the external validity, the study was performed on a single, large development project. Therefore, there is a risk of single group threats, which applies when the result looks at a single group. More empirical studies are needed to generalize our result. The size of the code base and the development organization are at a much larger scale than many commercial products. Therefore, it may be that smaller projects may not show similar trends. We also cannot generalize our results to software using other languages or platforms. More replication studies are needed for other types of development projects.

Our data spans almost 1.5 years of development work. The time scale is large enough to compensate for any unusual events that may skew the result. If there were any special events that may have influenced the result, it is unlikely to have contributed to a significant deviation of our data.

Finally, Type I error occurs “when a statistical test has indicated a pattern or relationship even if there actually is no real pattern” [BSL99]. Type 2 error occurs “when a statistical test has not indicated a pattern or relationship even if there is a real pattern” [BSL99]. Our use of statistics was to support data mining. As such, an entire population was used as discussed in the introduction section. Type I and Type II errors become a concern if a random selection of samples were used from our population. Because we used the entire population from our data, arriving at a wrong conclusion based on random

selection of samples does not exist in our case study. We also supported our result using data splitting.

Basili et al. stated that drawing a conclusion from one empirical study in software engineering is very difficult, because any number of context variables could have influenced the result [BSL99]. For this reason, we cannot assume that results from this project can be generalized beyond a similar project. One criticism of empirical studies is that the result may seem obvious after the fact, but this is a misguided belief as some important facts were discovered and/or reconfirmed through the evidence collected. Furthermore, we need to perform more empirical research and replication studies if we are to gain a better understanding of software development practices.

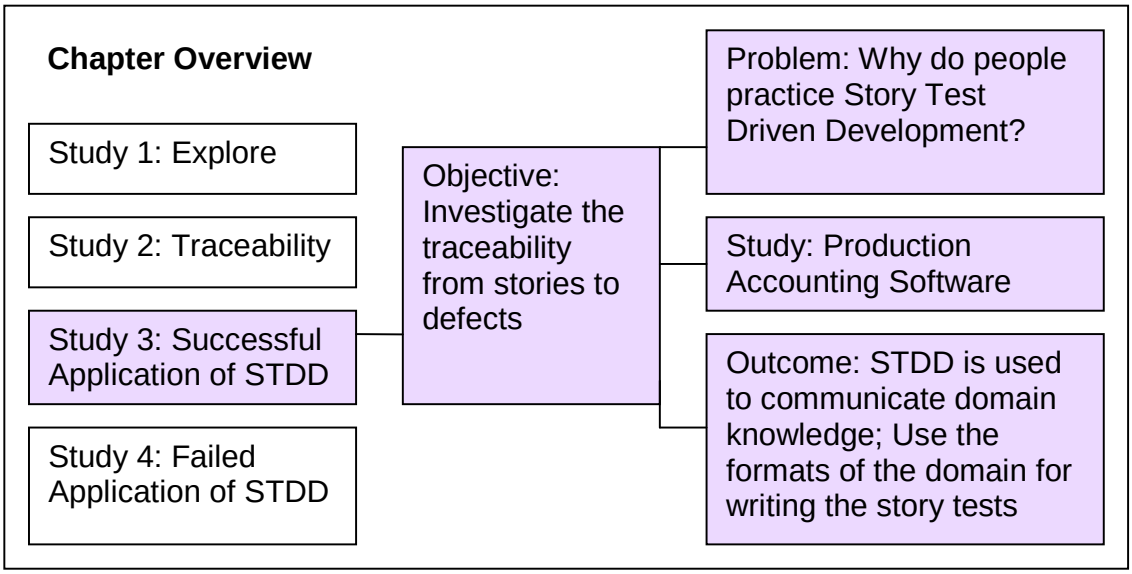
A correlation analysis cannot define the causality of the relationship without further experiments. Therefore, we invite other researchers to validate our results with additional project data. Our empirical study can contribute towards a better understanding about collecting measurable attributes for controlling and monitoring requirements-related defects. We also discovered that data collected by tools, such as Jazz, can provide a good basis to point to potential requirements defects and can rationalize decisions on where to spend inspection and testing effort.

5.7 Summary

Our study suggests that the number of related stories and the number of indirect stakeholders are very important in the relationship between stories and defects. In addition, we discovered that one's own knowledge of who works on what work items that is immediately related to one's own work is more important than one's breadth of

knowledge on what the entire team is doing. Our study suggests that we may need to look further into how people communicate between team members, especially on one's knowledge about who is working on what. What the result suggests is that perhaps story tests are best at making these two attributes much more noticeable throughout the development progress. If story testing try to minimize defects at the end by linking stories, code and the tests, perhaps it is trying to solve the complexity of discovering related stories and identifying indirect stakeholders as early as it can. The other attributes did not have any correlation when linked them to stories and defects. This particular study is only meant to provide insights and not to be used as a definite answer to the overall research question that we are trying to solve.

CHAPTER 6: A CASE STUDY OF SUCCESSFUL PRACTICE



6.1 Problem Statement[8]

We wanted to find an Agile team that believes that they absolutely cannot work without Story Test Driven Development and also successfully adopted and practiced Story Test Driven Development for the project. We need to discover aspects about the team and the project that are fundamentally different enough that simply introducing a tool such as Jazz cannot replace Story Test Driven Development. By observing their project and the team, we want to find out why they stuck to Story Test Driven Development even though others did not.

This Chapter describes the case study done on a company that practiced Story Test Driven Development successfully. We define successful adoption to mean that Story Test Driven Development is an integral part of their development process and the

[8] This Chapter appeared in the following paper: Park, S., Maurer, F., Communicating Requirements Domain Knowledge in Executable Acceptance Test Driven Development, Proc. 10th International Conference on Agile Processes and eXtreme Programming, Pula, Sardinia, Italy, pp. 23-32. The copyright release form is attached in Appendix II.

company practiced Story Test Driven Development for the entire duration of their development project. Moreover, the team embraces Story Test Driven Development as an integral part of their development process. We analyzed our observation about the team and interpreted what aspect of Story Test Driven Development was the key to their success. In doing so, we can find real benefits and real issues involved in practicing Story Test Driven Development.

6.2 Research Design

In this section, we describe our research methodology and our research design. Human factors are difficult to measure because it is impossible to measure the software engineering process independently of the practitioners. When we investigate a real-life software development project, we can't always measure, control or identify all of the factors that influence the development process [RPT+08]. Therefore, to facilitate our qualitative research in such settings, we decided to leverage a case study research strategy, Our case study is done at a local company, which we will refer in the dissertation as company C. While we were collaborating with company C for an extended period of time [Y03], the fieldwork for the current study was done over two additional days during which we conducted in depth interviews with members of the company C team. Our results are based on newly collected data as well as insights from previous collaboration. The new data represents over four years of development practice. We interviewed three additional software developers and one project manager for detailed data collection. In addition, we interacted with five additional software developers to corroborate our findings. We did not record the interviews, because recording the interviews was seen as

to be too intrusive in the company environment. However, any interesting remarks made by the developers were carefully written down during the interviews along with any observation we made. The length of the interview varied greatly depending on how much information the developers were providing for our research. We also participated in one daily scrum meeting and observed three other daily scrum meetings. Our empirical data also includes direct observation of a developer who was engaged in a debugging process of a failing story test, which lasted about one hour. We wanted to understand how story tests were used to fix the failing code.

Our analysis of the collected data involved open coding and code categorization using our field notes [GM07]. During open coding, we identified a set of codes that could provide most insights into the data from our field notes. Then we categorized the codes to determine the relationships among the identified codes and a list of themes was generated.

6.3 The PAS Project

A case study assumes that contextual condition is important in the phenomenon under study. Therefore, in this section, we are going to briefly describe the PAS development project. PAS is production accounting software for the petroleum industry. The purpose of oil and gas production accounting software is to keep track of oil and gas production and calculate various capital interests invested in the oil and gas wells. Company C already had an existing production accounting software called Triangle, but the system needed to be rewritten due to the obsolescence of technologies used for its development.

The PAS project was sponsored by four of the largest oil and gas companies in Canada. An oil and gas production accounting system is an extremely critical software system for oil and gas exploration companies, because engineers and production accountants not only have to keep track of their oil and gas productions, but they need to be able to calculate tax and various interests[9] being represented in their oil and gas wells. Because each country and province has their own unique set of regulations on tax and interest calculations, it is important for the oil and gas companies to use software that reflects the regulations for the political jurisdiction where the well exists. The engineering and accounting knowledge involved in the petroleum industry in order to build PAS is so complex that it is absolutely impossible to build the software without having someone with the expertise who can lay out the information properly to the software developers. While all software development requires business domain experts, the problem with PAS is that production accountants need years of training before they understand the domain. The knowledge is not easy to be picked up by developers on the side. The team also needs someone who can keep track of the changing set of government regulations in the oil and gas industry.

The number of software developers in PAS project fluctuated over the last four years of its development; therefore, the amount of knowledge about the project development within the team fluctuated. At the time of the fieldwork, PAS is already deployed and operational in the client's work environment. Our contacts in the company told us that there are about 80 software developers, testers and clerks at the time of the

[9] Interest: how costs and revenues are shared by stakeholders

fieldwork. For an Agile development team, it is quite a large team. The team is split up into several subteams (one for each major component).

The development area is a large open space. Each subteam had, what they called, a SPOC (Single point of contact) and an SME (subject matter expert). A SPOC communicates the progress of the team and addresses any concerns that other teams might have on the component they are building. The SME is the person who has the domain expertise to define the requirements, answer any domain-related questions from the developers and test the end products to ensure that the requirements were correctly understood and implemented by the developers.

Each subteam had 2 to 8 developers and they stay as a team only for the duration of building the specific business component. Each team holds a daily scrum meeting in the morning. Due to the size of the team, each team had separate scrum meetings and each team sent a team representative to inter-team daily scrum meetings. In addition, they kept track of bug lists using Jira [Ji11]. There were 927 ‘GUI Smoketests’ that test the user interface layer, 93 report regression tests and numerous unit tests and other types of tests that we did not look into carefully for this research.

6.4 Observation

In this section, we present our observations by describing the company C’s story test-driven development process. We first describe our observation and then we summarize the implications of our findings.

6.3.1 Choose the Requirements Specification Tool from the Customer's Domain

The requirements specification is defined using the language and formats of the business domain. This can reduce the extra overhead of learning the specification tool by the domain experts (customer representatives). Our case study shows that the domain experts chose Microsoft Excel as their requirements specification tool, which is a standard tool for communicating production accounting data in the oil and gas industry. The standards for the data formatting are regulated by the provincial energy regulatory boards [ER11] and production accountants generally use Microsoft Excel to facilitate their business communication. Although production accountants are not required to use Microsoft Excel, it is a common practice to do so in the oil and gas industry.

Microsoft Excel became a preferred tool for requirements specification, because Excel was familiar to the domain experts. Excel has features such as Visual Basic macro programming and pivot tables. The domain experts understood and used these Microsoft Excel features proficiently. A story test file has 12 macros, which are used to create these table templates and help with the test automation process.

One Excel file contained many requirements and the developers considered one Excel file as one story test. Each story test is composed of many calculation worksheets, which are represented using Excel worksheets. Each worksheet can contain 20 to 650 rows of test data and about 3 to 30 columns. Each row can return more than one expected output result.

There are a total of 17 business components in the PAS project. Each business component had multiple story tests and each story tests had multiple worksheets (or the developers called them 'views'). There are a total of 321 story tests for the PAS project.

The domain experts were able to write and maintain these tests whenever a feature is added or changed in the software.

The use of Excel was motivated by the ease of transferring and codifying the domain knowledge via this medium by the domain experts, especially because Excel is the conventional tool in the business domain. It is easier to codify tacit knowledge to explicit knowledge if the tool is already utilized to facilitate communication in the customer's domain. In addition, (1) domain experts who are familiar with the specification tool are more likely to create and maintain story tests for the developers and (2) the tool can communicate the domain knowledge best because it can represent the domain data appropriately. What makes Microsoft Excel interesting is that this tool is universally well known and easy to learn by both the customers and developers. Finding a common tool that can be used and understood by business experts as well as the development team is a crucial and important condition for a successful STDD process. Excel not only allowed the production accountants to leverage their existing computer skills for writing the requirements, but it also provided the contents in a form that developers can easily turn into story tests.

6.3.2 Communicating the Business Domain Knowledge

In this section, we are going to analyze the kind of knowledge their story tests are conveying to the developers. Table 8 shows only a very small snapshot of a large story test that has 644 rows and 14 columns, which is a typical example set of scenarios required to test for real-life production accounting. Table 8 is testing for a contract allocation. The last three columns show the sum of the expected share of the energy for

everyone who has a stake in the reserves profit. We found that most of the tests are transaction style calculations. Generally the tests identify *where (a physical entity* such as reserves or a facility), *who* the transaction is for, *what* values should be assigned and optionally *when* the transaction occurs. The format is typical of any production accounting spreadsheets. The test data is created by the domain experts using similar production data, but the data is so realistic that it could be the real production accounting data.

In the following paragraph, we will explain how this table is used as story tests. These spreadsheets are test values. Therefore, the spreadsheet may serve many stories and it may contain many features that will be developed in software. Therefore, there is no one-to-one relationship between stories and these examples. Like Fit tables, the spreadsheet shows the input values and output values. It also contains formulas on how these numbers are derived.

Therefore, for example, if the developer needs to write a function that produces "Sum of PRICE", he would refer to this spreadsheet example, analyze how the inputs are translated into the final output value. The developer would then write the automated story tests that extracts input and output values out of the spreadsheet automatically. Much like Test Driven Development, he needs to setup the test fixtures before he writes the code. After he writes the actual code, he would test his code against the story test that he wrote before. The customers are told how to to execute these story tests. Because the values are automatically extracted from the spreadsheet, the customers can now change the values in the spreadsheet to test whether the code written by the developer produces the right results even with different numbers.

Table 8: An Example Snapshot of Story Test Definition

Fac ID	Cont Name	Party Name	Prod Code	Cont Type	Settlement	Sum of VOL	Sum of PRICE	Sum of VAL
100040500 722W500	royalty C5	Farmer Cassie	5-SP	Royalty	ash	.76	339.98	938.37
					ash	.76	339.98	938.37
	TIK on gas – 100%	Kathy and Co.	AS	Royalty	IK	.2	235.69	518.51
					IK	.2	235.69	518.51

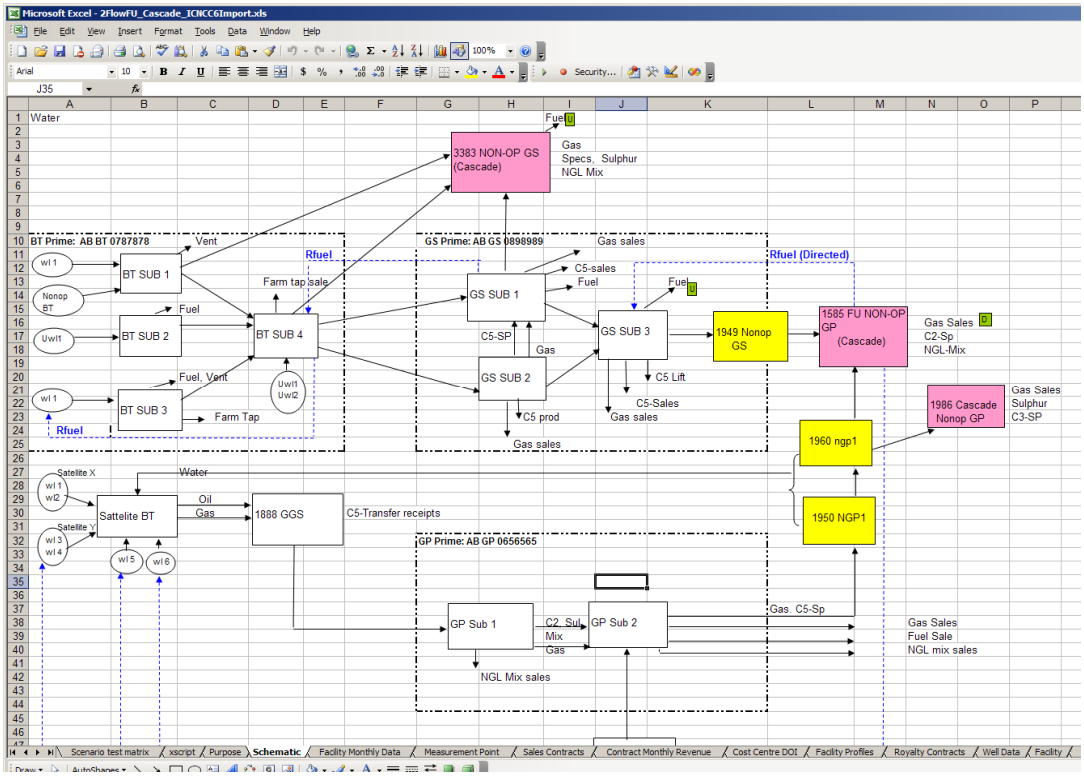


Figure 5: A diagram explaining the business process involved in a battery facility

The domain experts also provided workflow diagrams that are not executable, but they complement the executable specification. They were designed to inform the developers about the business workflow for the story test. Figure 5 shows a workflow

diagram of a battery facility[10]. Our analysis shows that the domain experts need to provide two types of documents: testable requirements specifications and an overview document to put the specification into context. The overview document can be used by the developers to point at something to ask for more information about the domain.

Making the Requirements Specification Executable

The testing framework is based on Excel, Ruby and JUnit. Ruby is used to automate the user interface layer testing. JUnit is used to execute the script file and to compare the test output values. Executing one story test can take up to 1 hour or more, because it simulates real-life production data and calculations and it runs against the UI layer. A production data file contains months or even years of production data. The largest Excel file is about 32 megabytes in size.

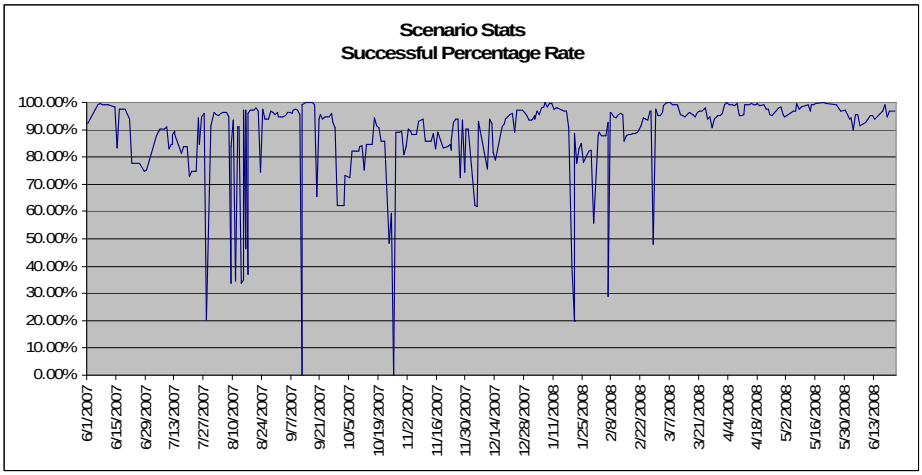


Figure 6: A time-series graph showing the percentage of story tests succeeding at the end of each sprint. 0% success rate was due to a test automation problem at the time rather than any serious software malfunction.

[10] Battery facility: a plant where raw petroleum is separated into different types of hydrocarbons

Figure 6 shows a time-series analysis graph of percentages of succeeding story tests captured at the end of each sprint. Notice that over time, the developers became much more conscience about passing the story tests. But we cannot make a definite conclusion about the quality of the software from the graph.

We wanted to get a better understanding whether the graph reflected the amount of domain knowledge that was transferred to the developers. To do so, we asked developers to explain what happened in the development stage by looking at some of these points in the graphs. We asked the developers to explain specific parts of the story test that were failing. They explained it in terms of how they can fix the problem technically, but they could not explain the domain knowledge behind these tests by putting it into context of the industry. However, they knew enough to explain why the test would fail for the specific story test that they worked on. Based on our interview data, we can assume that the team's understanding of the business domain is fragmented across many developers. It also meant the story tests provide enough information for the developers to implement the code even if they have limited understanding about the domain in which it will be used. The regression tests using story tests were important because they highlighted this knowledge gap among the developers. Different teams were responsible for developing different parts of the workflow process. Therefore, these story tests made the knowledge gap transparent to everyone, because it allows them to measure what story tests they understood and passed. More importantly, it allowed them to identify who they should talk to for specific story tests (requirements and domain knowledge). Therefore, they knew who and when to seek out help when these tests failed.

Unlike unit tests, failure in story tests meant they misunderstood domain knowledge, thus it signalled possible problems in delivering business value to the customer.

They did not need in-depth production accounting training to fix the software problem, because executable acceptance tests usually gave enough information to identify the problematic code and also provided the expected answer. However, a more in-depth empirical study is required to understand the developer's cognitive processes involved in going from failing executable acceptance tests to fixing the code.

6.5 Discussion

We discovered that the purpose of having story tests is to *communicate* the domain knowledge to the software developers that are pertinent in understanding the software requirements. The story tests were a feedback system for the developers to confirm their understanding about the requirements and the business domain. Previous research also validates our finding [M07, MMC06]. More than any other types of development artifacts, story tests are utilized to fill the knowledge gap between the domain experts and the software developers. The failing tests from the automated regression tests are a good starting point for developers to question their understanding about the domain. Without such feedback system, the developers would not know how to validate their understanding about the requirements. However, no one previously looked at how specific types of domain knowledge are written in executable acceptance tests.

The *medium* used for acceptance test specification is important for an successful STDD process. Previous papers found that tools are important [MMC06]. However, we discovered that tools are important for the domain experts more than the developers. The

team discovered that production accounting knowledge is very well organized using Microsoft Excel. We hypothesize that Microsoft Excel made the test specification easier for the domain experts. By giving more power to the domain experts, the developers were able to gain valuable acceptance tests that became critical artifacts for their success. We believe that directly utilizing the language and formalism of the business domain (instead of development oriented languages and formalisms) will improve communication between the business side and the development side of a software project.

We also discovered that the *context* is also important. First, we discovered that the acceptance tests not only provided testable example data, but we also need to provide overview documentation about the domain knowledge and business workflow diagrams. The extra information helped communicate the necessary domain knowledge needed to understand the acceptance test specifications.

6.6 Threats to Validity

For external validity, we believe our case study can be generalized to very large software development projects where the software developers do not have complete understanding of the domain knowledge. However, our case study only supports transactional style domain data. For threats to internal validity, the observation was collected and analyzed by the author of the thesis, which may have introduced some unintended bias. Our interview data also mostly reflect the perspectives of the developers. To ensure construct validity, we performed a detailed inspection of the tool and the requirements. We are confident about our findings, because our observation and test data represents over four years of development practice.

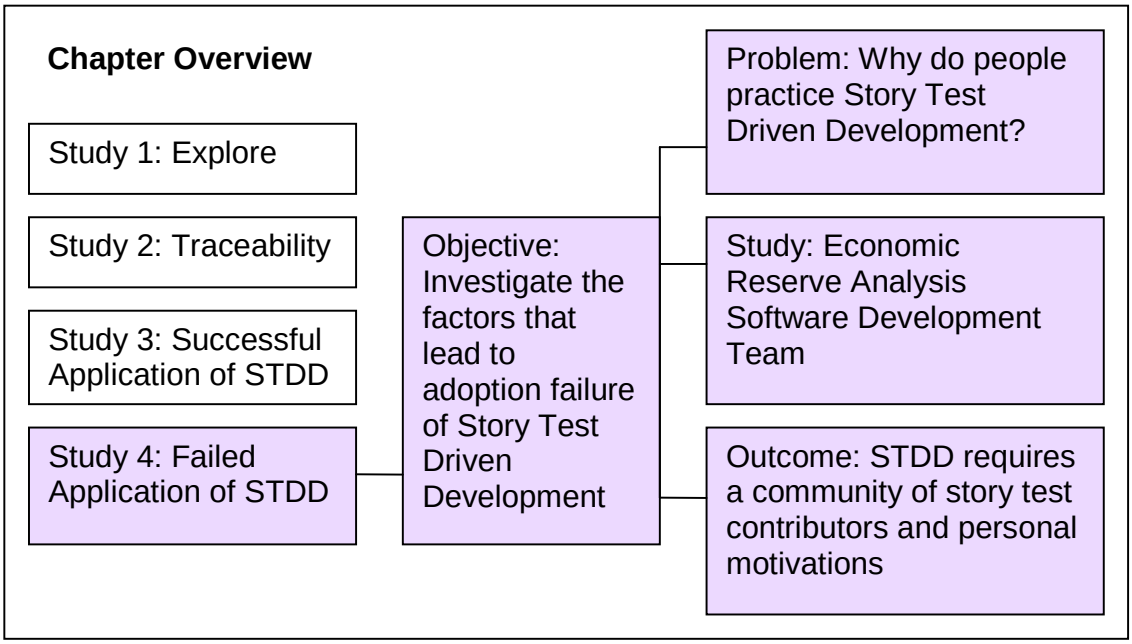
6.7 Summary

Our study shows that the main reason for the success of Story Test Driven Development is to communicate domain knowledge that is pertinent in understanding the software requirements. Knowledge held by the domain experts was not something that developers can easily learn through simple questions and answers with customers. It is also not something that can easily be communicated with simple stories. The story tests were part of a learning tool for the developers. The story tests were important in making sure that the software developers understood all of the different scenarios that exist, so they can implement domain knowledge into software functionalities. The automation of the story tests against the code was to make sure that no other developers broke the existing functionalities and end up returning different results. Not all of the developers need to become experts in production accounting to build software for production accounting. However, all of the developers must ensure that the story tests were passing after their implementation was added. Even if they do not know all of domain knowledge in other aspects of the software, passing the entire story tests meant that they know their code did not mistakenly negatively impact any other parts of the code. The story tests were an important backbone of communicating the correct domain knowledge to the developers without too much training cost. Therefore, we believe the key aspect that Story Test Driven Development actually serves in Agile context is communicating the domain knowledge to the developers.

In addition, what worked so well for this team was that the domain experts used the tools that they were familiar with – Microsoft Excel. The customers did not have to

learn a new tool or worry about the test automations. The developers were the ones who figured out how to extract data from the examples that were provided by the domain experts and automated them for their development needs. Therefore, there is no additional overhead for the customers to participate in the Story Test Driven Development. Based on this research, we propose example-driven story testing – using the examples of the domain for story testing for the purpose of teaching, communicating and learning about the domain specific knowledge. In the next Chapter, we explore the obstacles in eliciting examples from the customers.

CHAPTER 7: WHAT IS THE BIGGEST OBSTACLE? A CASE STUDY



7.1 Introduction

In Chapter 6, we discovered that Story Test Driven Development is particularly useful for communicating domain knowledge between customers (domain experts) and the developers. The automated testing aspect of the story tests allows developers to implicitly learn and directly test their knowledge about their understanding of the domain knowledge and find out how it is implemented in software. Story tests were not a quality assurance tool, but a validation tool about how domain knowledge should be implemented in software. Especially in a large software team such as the team in Chapter 6, the automated story tests are a vital part of validating everyone's implementation.

We also discovered that using the tools and formats of the domain is important in their success. Knowledge transfer is something that no other Agile techniques can replace and it is a purpose that only example-driven story testing can fulfill. Story Tests are used particularly for the purpose of transferring and communicating domain knowledge to the

team members. As mentioned in Chapter 6, we call this use of Story Test Driven Development "*Example Driven Story Testing"*. However, could a team also encounter a problem with example driven story testing? If so, what could it be?

In the following, we present a case study of a team that also used Example-driven Story Test Driven Development in a very similar way as the team in Chapter 6, but STDD did not work out for the team. The team in this Chapter is also building software that is similar to the team in Chapter 6, but the team is much smaller. The organizational arrangement is different than the team in Chapter 6. The team also brings interesting insights into the way Example-based Story Test Driven Development works, because it allows the researcher to see the problems better than only examining successful cases. As mentioned in Chapter 6, we cannot make any definite conclusion based on one case study, but a single case study can provide very rich insights into the problem under study. As mentioned in Chapter 3, qualitative research is not always about finding the definite laws of the universe, but to provide insights into the phenomenon under study. The case study in Chapter 7 is presented for a comparison purpose in order to gather more rich evidence about Example-driven story testing.

7.2 Background

This section describes the background of the team and the project that we used for our case study. The purpose of this section is to provide the context in which the case study is done. A case study assumes that contextual condition is important in the phenomenon under study. Therefore, in this section, we are going to briefly describe their software development project.

This case study is done at a small local software company that is building software for oil and gas industry. The case study was done in March 2008. Unlike the team in Chapter 6, the company and the employees want to stay anonymous for our research, especially considering this is a failed attempt. Software they were building was petroleum reserves economic risk analysis software. Their practice of Story Test Driven Development came to an abrupt end due to personnel shifts and unforeseen reorganization of the company. We will explain further in the chapter, but their main reason for the failure is the lack of ownership and the lack of community of story tests contributors. However, they were able to practice Story Test Driven Development for roughly about 6 months (although the project lasted two or three years) and this case study covers their practice during that short time period.

At the end, we interviewed the team to gather their retrospectives on the project. We recorded the interviews and any interesting remarks made by the developers were carefully written down during the interviews along with any observation we made. Each interview lasted about 2 hours. Our analysis of the collected data involved open coding and code categorization using our field notes [SC98][11]. During open coding, we identified a set of codes that could provide most insights into the data from our field notes. Then we categorized the codes to determine the relationships among the identified codes and a list of themes was generated.

Their software is economic risk analysis software for petroleum reserves. The purpose of economic risk analysis software is to analyze whether development of chosen

[11] The two transcripts that were recorded and transcribed are available in the Appendix III in verbatim for the examination purposes only as well as their codes. In accordance with the conditions of the ethics approval, the transcripts are not included in the final version of the dissertation to protect the privacy of the people we interviewed.

reserve is economical enough to purchase, drill or extract oil and gas from either proven or unproven reserves. Unlike the team in Chapter 6, the software they were building is interested in future viability of a well rather than accounting what is already produced. Their software allows the engineers to predict how profitable the reserves will become given production rates and capital interests to make the reserve active. Software may also be used for the purpose of predicting profitable reserves for the exploration purposes as well. Much like the case study in Chapter 6, this company also has an existing product that is already being licensed to their clients. However, the team was trying to add additional experimental functionalities to enhance the prediction capability of their software.

The team that practiced Story Test Driven Development was a five-member sub-team within the organization with about 30+ members (although the original team was three people). The company was not practicing Agile methodologies and the management did not believe in its usefulness. However, this particular sub-team was practicing Agile methodologies within their team and was trying to spread its usage to the rest of the company. Therefore, the main difference between the team in Chapter 6 and Chapter 7 is that the team in Chapter 7 is also battling the overall Agile adoption as well.

The developers did not know about *Story Test Driven Development* (or any of the many other names it is called), because they did not know about its existence. They felt that this was uniquely their invention. However, their *invention* was remarkably similar to how the team in Chapter 6 was practicing, which is why this particular team made an interesting case study. We did not tell them about the existence of story testing in order to not interfere with their view of how the team should practice their version of story testing.

An economic prediction software is an extremely critical software system for oil and gas exploration companies, because engineers not only need to keep track of their past production rate, but also use them in conjunction with financial data, government tax rates and other various capital interests[12] that are reflected on the particular reserves. In addition, because each country and province has their own unique set of regulations on tax and interest calculations, it is important for the oil and gas companies to use software that reflects the regulations for the political jurisdiction where the well exists. The amount of engineering and financial knowledge involved in the petroleum industry in order to build economic risk analysis software is so complex that it is absolutely impossible to build such software without having someone with the expertise who can lay out the information properly to the software developers. While all software development requires business domain experts, the problem with economic reserves is that developers would need years of training before they understand the domain. Like the project in Chapter 6, the knowledge is not easy to be picked up by developers on the side. The team also needs someone who can keep track of the changing set of government regulations, the latest news on oil and gas plays[13] and production rates of nearby reserves.

The team under study is made up of four developers and one project manager. This particular organization put their employees into teams based on their job functionalities. For example, developers all belong to the developer team and the project managers belong to the project manager team. These developers and project managers are assigned to the project only for the duration of a particular assignment and then they are broken up again. Therefore, the five member team is only going to work together for the

[12] Interest: how costs and revenues are shared by stakeholders
[13] Buying and selling of reserves

duration of their project. The project manager may even have to manage multiple projects at a time if they were lacking project managers and even share the responsibilities with someone else. It is also possible that the project manager may be reassigned to another project in the middle of the project. Based on the interviews, it did not seem like the project managers had much say in what project they want to work on. The developers, however, usually stayed with their project until it is finished or abandoned. One developer mentioned how he was happy to be assigned to work with a particular person, but regretted that he could not work with the person again in the next assignment. It also meant there is a testing team where the project gets transferred to after the development is done.

The domain knowledge would have to come from the business analyst team (sometimes also called Project champions but developers and project managers used different terminologies for them) or the project manager team depending on who had the expert knowledge. It is often just a guess as to who may have the domain knowledge, but usually the developers knew who would have the domain knowledge (or even have preferences on who was better). But generally, the team can only get one business analyst. Not every business analysts and project managers were hired with the domain knowledge. Therefore, one developer mentioned if they were assigned a project manager without the domain knowledge, it was really painful to work together. In addition, they had to find the domain experts outside of the project team. The developers mentioned that asking someone for their knowledge was not always an easy request, because these people may be juggling multiple projects. If the project is not assigned to an analysts' list of projects to manage and analyze, the requests do not always get handled efficiently. Project

managers sometimes fulfilled the role of analysts, but analysts did not manage projects. In addition, if the person is managing multiple projects, projects that are not high on their priority list may fall short of help. In situations where the right kind of domain experts is not assigned to their projects, they mentioned that asking the knowledge from the fellow developers may be easier. Some developers already had decades of experience in making this type of software and these developers knew enough to be domain experts in some areas. Sometimes asking fellow developers might just be the faster way to get the necessary domain knowledge.

7.3 Observation

In this section, we present our observations by describing the pitfalls of how the team practiced story test-driven development process. We first describe our observation and then we summarize the implications of our findings.

7.3.1 Ownership of the Story Tests

As mentioned in Chapter 6, the standard tool for communicating production knowledge is Microsoft Excel. It is also reflected in this company and the team used Microsoft Excel spreadsheets for writing their story tests. The developers created Microsoft Excel spreadsheets with the necessary formulas. One developer was particularly proud of his work on how he managed to automate the nightly regression tests based on this Microsoft Excel spreadsheet.

He said:

Excel is a cell-based calculator. So it's extremely flexible. It doesn't have any structure. The calculations that we want are, ah, basically time based calculations, and, not only time-based variable, time-based variable calculations. So you want to be able to have variables and relationships between variables, but not cells. And Excel, you know, you have to repeat formulas. You have poor data handling.

Therefore, the developer created a template in which the numbers would be generated randomly by Microsoft Excel and were feed into the automated tests. The developers asked the business analyst to fill in the spreadsheet with some examples and formulas. For example, one of the developers said, "what we found through this was it was a really good way to get business people to express requirements in a concrete way and that was a big challenge." However, despite the usage of the domain tools and the automation tests based on the Excel spreadsheets, eventually only the developer who invented the tool ended up using it.

The problem was the ownership. The benefit of using the domain tool is to allow the customers (the business analysts or the project managers in this case) to explicitly write out their knowledge in a form and with the tool that they were familiar with. However, in this case, the spreadsheet had specific rows and columns where a specific type of information must go into. The template was created by the developer, not by the domain experts. Although not spoken, the impression we got was that the ownership of the spreadsheet was that of one particularly person, the developer who invented it, and neither the team nor the domain experts.

In addition, there was a lack of freedom in how much more information the domain experts can provide based on the rigid templates provided by the developer. If the domain experts provided more information, then the developers had to automate them, which would add more time to already busy schedules. However, the developer mentioned how it would not be too hard to accommodate adding additional lines in the automation code to handle extra rows and columns. One of the participants said, “it turned out that, ah, the business person was changing the Excel spreadsheets and constantly saying that they were done. You know, within a day or two days, changing them again and trying to sneek it in and. Ah, the automated tests will catch the stuffs were failing because things have been changed in the spreadsheet.” In addition, if they added additional lines into the story tests and the regression tests failed, the domain experts would have cranky developers the next day. In other words, the story tests belonged to the developers. The domain experts did not have the ownership of these artefacts. They were perceived to be merely guest contributors.

However, eventually the ownership problem propagated even to the developers. The main problem was the other developers did not really know how data was laid out on the story tests and were just afraid that they would break something. In addition, there was no real advantage for the developers to spend their time to work on these story tests when they had a pretty good set of automated tests based on unit tests. The developers did not really feel the need to write out their knowledge in both unit tests and story tests. If the business analysts were not going to update these story tests, then they also did not see the point of updating these story tests themselves.

Therefore, it is not only important to use the tools and format of the domain, but also that the customers (domain experts) must own these story tests. However, in order for the domain experts to have the ownership of the story tests, the story tests need to be flexible enough for them to write, change and maintain them comfortably using their knowledge and background.

7.3.2 Community of Contributors

Writing and maintaining the story tests take time. The project manager said that he simply did not have the time to write the story tests. He was juggling many projects. He was reassigned to another project before the end of the project and another project manager was substituted into the project. The previous project manager said that he liked the idea of story tests, but he said that he did not have the time to help the developers to write them. The project manager did not foresee his role to be a permanent part of the project. Therefore, he wanted the developers to be self-sufficient. Leaving behind story tests would mean that he would needed to be contacted even after he left the project.

A developer said that the idea of story testing met its demise after the first project manager was reassigned to another project. He said, “now after the project got going a while, we switched business analyst to somebody else who wasn’t quite as detail oriented and not as good at expressing requirements and, you know to the level of details that you could nail it down and actually program something to do it.” With the new analyst, it was hard to pick up the momentum again. The developer asked a business analyst, but she was usually too busy to get an answer. What the project lacked was the person who could fill the domain expert position.

In addition, the business analysts and project managers were not evaluated based on writing and maintaining the story tests. The project manager said that the developer who built the story testing tool was working in this industry for decades along with another developer. He believed that the developers had the knowledge to write software themselves. He hoped that they would write and maintain the story tests and just ask him for more information when they needed a little bit of additional information. Because story testing was entirely the developers' initiative and their invention, the domain experts still saw story testing as developers' tasks, not the customers' tasks. Story testing was not seen as something they would get personally rewarded. In our view, what lacked in this company was a community of contributors who can maintain these story tests regardless of the team and the projects. It was viewed as a task that was assigned to a specific person and people did not want extra experimental task on their to-do list, especially when the company was not fully onboard the idea of story testing.

7.4 Discussion

In Chapter 6, we described that the purpose of having story is to *communicate* the domain knowledge to the software developers for software functionalities. The story tests were a feedback system for the developers to confirm their understanding about the requirements and the business domain. However, we discovered that practicing successful story testing requires additional factors that we did not readily see from the study in Chapter 6 alone. Simply using the tools of the domain does not guarantee success. The study in Chapter 7 shows that ownership and community matters in its overall success as well.

The tools and formats both need to be organized by the domain experts, because it gives the customers the feeling of ownership of these story tests. Asking the domain experts to simply fill out existing templates or forms does not foster the sense of ownership by the domain experts. Perhaps the format of the story tests need to be discussed by both the developers and the domain experts to figure out what formats and tools may help both parties.

In addition, we discovered that there needs to be a community of contributors for the story tests. In an organization where you do not know how long you will end up working on the project, the project managers and business analysts were not easily accepting new responsibilities that may tie them to the project. Unless the entire organization was practicing Story Test Driven Development, like we have seen in the team in Chapter 6, the domain experts were not ready to spend time implementing story tests for a single project that they did not even know how long they will eventually work on. The team also felt that the practice will not last once they move onto the next project. In the end, Story Test Driven Development is a practice that requires a community of contributors. It is not just a one person assignment.

In addition, an alternative answer to why the team failed to practice Story Test Driven Development is the lack of a process in using story tests. We suspect that the process did not emerge like the other teams, because the customers were not motivated to engage in the process and the lack of community that can provide the story tests.

7.5 Threats to Validity

For external validity, we believe our case study can be generalized to software development projects where the software developers do not have complete understanding of the domain knowledge. However, our case study again only supports transactional style domain data. For threats to internal validity, the observation was collected and analyzed mainly by me, which may have introduced some unintended bias. Our interview data also mostly reflect the perspectives of the developers. To ensure construct validity, we performed a detailed inspection of the tool and the requirements. The case study provides an additional case study for the comparison purpose in addition to Chapter 6.

7.6 Summary

In this study, we have presented another case study for the comparison purpose. This case study is a failed attempt at Story Test Driven Development despite using the tools of the domain for communicating domain knowledge to the developers. It shows that adoption of Story Test Driven Development requires customer ownership and a community of contributors.

CHAPTER 8: SYNTHESIS OF FINDINGS

The main goal for my research is to investigate why people use Story Test Driven Development in Agile software development. We explored three research questions in the dissertation. 1) What problems are faced by Agile teams in practicing Story Test Driven Development? 2) Investigate the relationship between stories, teams and defects. 3) What are the factors that lead to successful adoption of Story Test Driven Development? In Chapter 4 and 5, we discovered that the technical (programming/coding/testing) aspect of the software development is not the main usage of Story Test Driven Development. Chapter 6 and 7 presented two industry case studies of how Agile teams implemented Story Test Driven Development for communicating domain knowledge. In addition, we discovered that customers require not only a community of contributors.

Throughout all of our empirical evaluations, we addressed how and why Agile teams adopt Story Test Driven Development and what works in real life settings. We synthesize the findings and make generalizations on the uses of Story Test Driven Development. We synthesize the findings based on the empirical evidence from the studies and corroborate our findings with existing literatures.

8.1 Main Themes

We identified the following four main themes emerging from our empirical evidence.

- **Examples of the Domain:** Our observation suggests that story tests should be the examples from the domain. It encourages the domain experts to provide examples in the format that is comfortable to them.

- **Story Tests as Knowledge Repository:** We observed that the developers use the story tests to learn and test their domain knowledge by testing their code against the story tests.
- **Rewards and Motivations:** The customers require a personal reward in order to be motivated to provide these domain examples.
- **Community of Contributors:** Story tests should not be generated by one person. We discovered that it is easier to build the knowledge repository of story tests if there is a community of contributors. It spreads the burden of maintaining the story tests to the team rather than to one person.

In the following sections, we will summarize each of the main themes more thoroughly with supporting evidence from our studies. We also corroborate our findings with studies done by other researchers in the similar research areas.

8.2 Examples of the Domain

Story Test Driven Development is a communication technique rather than a software testing technique. In this section, we will discuss the potential adoption problems that a new technique, such as Story Test Driven Development, may need to overcome in order for Agile teams to adopt it into their development process effectively.

Why are some software engineering techniques adopted more readily than others? Story Test Driven Development is an innovation within the tools and techniques that make up Agile software engineering. According to the literature on Diffusion of Innovation (DOI), an adoption process is not an individual or an organizational decision,

but it needs to be analyzed from the community's perspective [FK93]. The benefits of the adoption must be evaluated in terms of the community, because the adoption usually depends on the size of the current and future adopters and it will have a strong impact on the inherent economic value of the innovation [FK93]. Widely accepted technologies will have faster innovation, more experts and better complementary and compatible tools. From these perspectives, Story Test Driven Development must solve problems that other techniques either cannot solve or are too costly to solve. It should not try to solve software engineering problems that other techniques can already solve at a significantly lower adoption overhead and cost. What Chapter 5 is suggesting is that there are other techniques and tools that are much more readily available than Story Test Driven Development that can solve specific software designs and testing problems that were discussed in the mailing list from Chapter 4.

In Chapter 6 and 7, we observed that Story Test Driven Development is used as a communication tool. The domain knowledge that the software developers need to learn is very complex. The story tests were used as a way to validate one's knowledge about their software implementation. Story tests are also a way to safeguard other people's code against unintended code changes from the new code, because developers only knew enough domain knowledge to build their own functionalities. Without the story tests, most people in the large software team would not know how their code changed other people's code and their functionalities.

Similar findings are observed in the previous studies [HH08, M08, OP09, HK06, SP04, ARS07, GBGP07, KNR09, CD07, PM08]. They stated that there is better communication with the stakeholders. [GHHW05, TKHD06] stated that they had better

understanding of what has been developed. [HH08, CHW01, St09, K06, HK06, R04, ARS07, MLSM04, ABL09] stated that they had better confidence about the progress and deliverables. However, our study adds to the existing body of knowledge that story testing is a process for communicating knowledge rather than a process to inform stakeholders about the development progress. It can solve the crucial bottleneck in communicating knowledge to the developers who do not have previous training in the domain.

We also observed that using the format of the domain worked well for the team in Chapter 6. A communication tool should embody as much of the essential information and business context as possible. There are two reasons for the use of the formats of the domain. The first reason is the motivation. Our research shows that the best way for domain experts to participate in Story Test Driven Development is if the domain experts use formats and tools of the domain. The story tests should be examples, possibly even examples right out of the domain as they are used. The domain experts and the end-users are familiar with these notations and the tools, thus there is less overhead in terms of providing the example for the purpose of story testing. Removing the barriers such as training cost for tools is important in order to encourage faster adoption.

Another reason is the communication of the business context. The story tests using examples of the domain provides better context for the developers to understand and learn about the domain. It should not be written and produced using software testing tools, because it not only loses the business context of the domain, but it is also harder for domain experts to help the developers if they have to use the tools and formats that are

not familiar to them. This finding is a new addition to the academic body of knowledge for Agile software engineering.

From the economical standpoint of the adoption diffusion, story testing needs to be looked at from both the developers' perspective as well as the customers' perspective. It is important to find the tools and techniques that can satisfy both of these groups with the least amount of initial training and cost. The use of software testing tools has a high training overhead for non-developers. On the other hand, we discovered that automating story tests is usually not too difficult for the developers once data becomes available. The bottleneck in the adoption process is the willingness of the customers to provide these story tests. We discovered that using the examples from the domain is a good way to alleviate this particular adoption problem.

The domain experts usually have their own standards of tools and formats that guide their discipline. These formats and tools have long traditions within the domain expert's field and sometimes there are very good reasons for their choice of formats and tools. Staying with the domain's standard tools and formats may encourage many domain experts to contribute, which also may help the developers in recruiting several domain experts to work together rather than having just one domain expert.

8.3 Story Tests as Knowledge Repository

The story tests can serve as the knowledge repository of the domain. Much the same way stories and a storyboard behave as information radiator for Agile teams [B99], the story tests are information radiator for domain knowledge. The examples used for

story tests bring contextual information into the discussion, which would help the developers understand better what the end-users want.

Cockburn states that a good information radiator should have the following attributes [C04]:

- Easily visible to the casual and interested observers
- Understandable at a glance
- Changes periodically
- Kept up to date

Many Agile studies emphasize the importance of the information radiator in the overall success of the project [RP08, EW06, Sh07, B08, SRSF06]. Sharp et al. state that much of the knowledge in Agile teams are tacit, except for the two tangible artefacts: story cards and the wall where the story cards are hung [SRSF06]. They discovered that the key mechanism for moving information is face-to-face interactions. The stories were artefacts for mediating, creating scaffolding, goal setting and coordinating resources. The information transformation occurred when these stories were turned into executable code. However, most of the transformation occurred implicitly between people, cards and the wall. In order for story tests to act as information radiators within this model for Agile communication, story tests also need to be the mediator and help with goal settings and coordinating resources.

Chapter 6 suggests that story tests do work in conjunction with stories and they also act as a type of information radiator. The story tests can provide additional information on top of the stories by providing information that is hard to convey with conversations only. For example, some of the difficult calculations are best

communicated with worksheets and verified using automated tests. However, much like stories and the wall, these story tests still require face-to-face communication in order for the experts to explain the main concepts behind the calculations.

In order for story tests to become the information radiator, it needs to be easily visible, understandable and kept updated, which is why using the examples of the domain is important. It is the universally common notation for both the customers and the developers. The visibility comes from the execution of the automated story tests. The results of the story tests against the code would provide a red or green light on how the information is being translated into software. Therefore, story tests can act as a much more concrete information radiator than stories because they provide an up-to-date status against the live code.

In addition, story tests should be seen as documentation for software. As seen in Chapter 6, when tests link the code to the examples, it not only provides documentation of how the calculation works, but also the business context in which software results will be used. Because of the traceability between the code and the examples through automated tests, story testing provides a great way to leave behind documentation about how the software works using examples and executable tests for validation. These story tests are valuable documentation artefacts about the software that customers can read and understand. Therefore, creating these story tests should be treated as a documentation and knowledge building process rather than tasks that only help the developers. In addition, the management should see the value in building such knowledge repositories, which will become valuable resources for the company even for other future uses.

Previous studies suggest that story testing can help everyone to understand quickly what has been developed [GHHW05, TKHD06]. Our studies suggest that story testing can further help the team by providing an information repository that can be valuable even after the software development is completed. It also suggests that Story Test Driven Development is meant for building knowledge-intensive type of software – ones that require a great deal of domain expertise knowledge and where software developers do not have much training in the domain.

8.4 Rewards and Motivation

The most influential theory on motivation is the Herzberg's two-factory theory of satisfaction and motivation [HMS59]. The theory states that employee satisfaction can be divided into intrinsic factors and extrinsic factors. The intrinsic factors are related to the work that is being done, such as recognition, achievement, responsibility, advancement, and personal growth. On the other hand, the extrinsic factors, such as company policies, supervisory practices, pay plans and working conditions do not have a high influence on motivations, because they are not directly related to the task at hand. Therefore, organizational changes that deal with extrinsic factors do not necessarily increase in the employee satisfaction.

In terms of the studies done in software engineering, Hall et al. published a study on what motivates software developers [HSB+08]. They identified 21 factors and categorized them into intrinsic and extrinsic factors. The intrinsic factors include identifying with the task, career paths, variety of work, recognition for work done, addressing development needs, technically challenging work, making a contribution,

trust/respect, equity and employee participation. The extrinsic factors were good management, sense of belonging, rewards and incentives, feedback, job security, good work/life balance, appropriate working conditions, successful company and sufficient resources. Extrinsic motivators are general working conditions and general good management, but these factors have less influence on the motivation of software developers. Therefore, the study concludes that task-based management may be better for software developers, because it itemizes the challenges into a list of problems that the developers must solve.

To the best of our knowledge, there is no research done on what motivates customers to participate in the software development process, particularly on Story Test Driven Development. The people who fill the role of the 'customers' in Agile teams may not be the employees of the company. These people also could come from diverse backgrounds. Therefore, it is difficult to categorize their motivating factors.

Based on Chapter 6 and 7, we only witnessed 3 of the 21 motivating factors from the domain experts. They are 1) identifying with the task 2) recognition and 3) sufficient resources. The other motivators were not readily observable. Borrowing the term from Hall et al.'s work, "identifying with the task" means having clear goals, having a personal interest in the problem, knowing the task's purpose and how they fit with the whole work. In addition, the worker needs to be able to produce an identifiable piece of quality work [HSB+08]. In Chapter 6, the domain experts had a clear role in the software development process. They also created and maintained the story tests. The artefacts that they produced, which are used as story tests, are clearly identifiable pieces of work. Their

work is also integrated into the whole software through the automated tests. Therefore, the impact of their work is clearly identifiable every time the automated tests are run.

We also observed that the domain experts require a clear set of measurable goals on what is needed of them, which was observable in both Chapter 6 and 7. Eg. How many story tests are required from them or how many examples are required? It allows the domain experts to count down how many they need to produce and gives them the motivation to finish their work. In addition, they would be rewarded with the feeling of accomplishments. However, having these goals was not enough as seen in Chapter 7. The domain experts knew what was expected of them, but they needed recognition for their work.

In Chapter 7, we observed the importance of recognition. Creating, collecting and maintaining story tests was not an organizational initiative. Therefore, the work that the domain experts had to put in for creating story tests was extra to their assigned tasks. From the domain experts' point of view, the story tests not only provided little to no personal reward, but it would become a hindrance to their other assigned tasks. The major hindrance was the lack of time. Therefore, the domain experts did not have sufficient resources to complete the story tests. While extrinsic factors such as having sufficient resources does not guarantee success, the study indicates that extrinsic factors can become a major hindrance for writing story tests.

Ahn et al. published a paper on the use of human computation for solving novel computer problems that require human brains [ALB06, AB05, AD08]. The finding of their work is that people are willing to contribute data for their study in exchange of gaining personal entertainment. The studies show that there is an intrinsic motivation for

non-employees to spend considerable amount of time providing data in exchange for personal rewards. These intrinsic factors are related to the need to satisfy their intellectual curiosity.

Given the right kind of rewards, we know that people will contribute as seen in Ahn's work. Our findings indicate that people will become an information contributor only if they can also become the information consumer. Information contribution is clearly a process that requires both give and take.

In addition, the rewards need to be given out systematically. Therefore, our study suggests that customers require direct and systematic reward for a set of story tests that they contribute. Our contribution to the academic body of knowledge is to view customer's involvement in Story Test Driven Development as an activity that requires personal rewards and motivations, rather than view it as an activity that is done for the good of the team. Even though the entire team will benefit from the technique, STDD is hard to execute without the proper rewards and motivations for the customers to participate.

8.5 Community of Contributors

According to Martin et al., there are eight types of customer practices [MBN09]: Customer Boot Camp, Customer's Apprentice, Customer Pairing, Programmer Holiday Support, Programmer On-Site, Roadshow, Big Picture Up Front, and Recalibration. Martin et al. discovered that customers participate in the process in varying degrees. For example, in Customer's Apprentice, the developers work on the customer team so that they can understand the complexity of the customer teams' role [MBN09]. In Big Picture

Up-Front, the business stakeholders are only involved during the envisioning process [MBN09]. In these situations, the customers are not directly involved in the development. Our studies suggest that this kind of relationship with the customers do not work very well for Story Test Driven Development.

In Customer Pairing, two members of the customer team work together to provide a “single-voice” to the developers. We have not looked into situations such as this, but we observed that Story Test Driven Development does provide a single-voice to the developers no matter how many customers are involved, because the story tests can only pass in one way.

In Customer Boot Camp, the customers are given a training event. We discovered that more training does not necessarily improve the overall adoption. It is better to accommodate the customers in a way that they can already reuse their existing knowledge and skills.

In Programmer Holiday, the customers are given a think-ahead time. This is a necessary condition in order for Story Test Driven Development to succeed, because the story tests need to be generated before the development begins.

In another Martin et al. paper [MBN09b], they discovered that customer team always exists in Agile teams, but their roles are different. These roles range from “Acceptance Tester” to “Political Advisor” and “Super-Secretary”. Because of these differences in their roles, we observed that it is important to create a community of contributors, especially using formats and tools of the domain. It is not enough to just assign one specialized person to be responsible for the task of writing the tests. The entire customer group needs to be aware of the story tests and it should be understandable to all

of them in such a way that anyone can participate in the Story Test Driven Development process.

In Chapter 6 and 7, we observed the importance of having a community of contributors. In the field of Open Source development research, Bekler [Be02] and Markus et al. performed research in terms of organizational innovation and virtual organization [MMA00]. One of the main characteristics of Open Source development is that there is no direct monetary compensation for their participation [BL01]. Therefore, the assumption is that their participation is based on altruism. However, some observed that altruism may not be the motivating factor. In some open source projects, the contributors are highly individualistic and seek to gain reputation, future career opportunities, peer recognition, better software and even financial rewards [FF01,HO02,T98].

We observed in Chapter 6 and 7 that having a community is an important factor, because the personal reward would not exist without the existence of the community. These types of rewards are different from monetary rewards given from supervisors where the rewards are negotiated ahead of time. As we discussed before, one of the personal rewards is a form of respect from the community, which cannot exist without the people who form the community. Bergquist and Ljunberg used the term Gift culture to describe such phenomenon in the open source community [BL01]. A gift culture exists in a community where gifts are given without obligation to repay. The motivation for the givers is to gain fame and respect from the community [BL01]. Therefore, having a community is an important pre-requisite for gaining the reward for the giver.

Another model is the Wikipedia type of contributions. Not everyone can participate in the Open Source community even if they want to, because the source code is usually controlled by a select few individuals. Therefore, the open source community is actually quite closed and much more structured than what most people believe. On the other hand, examples such as Wikipedia are truly open to everyone for contribution (although some may say there is less contribution these days). The wiki community also depends on having a community of contributors, but the limiting factors are contributors' skill sets and their motivations.

In the wiki community, researchers have observed behaviour that they call, "selfish altruism", which is based on the prisoner's dilemma. Prisoner's dilemma refers to situations where individuals seek to gain the most selfish payoff instead of cooperating with others in an event when there is no coordination or communication with the others. However, given repeated prisoner's dilemma situations, the game theorists observed that participants will choose the most optimal solution, which is cooperating with each other [A84, D89]. The cooperation only lasts as well long as the other side is also willing to cooperate. This model is referred to as "selfish altruism", because the cooperation only lasts as long as the participant is gaining the reward. Another model is "reciprocal altruism" where the participant will contribute with an expectation that they will receive their reward in the future [T79].

In Chapter 7, the original developer controlled how the story tests were generated and maintained. We observed that the domain experts were reluctant to contribute in this model. In this model, even though everyone, in theory, can contribute, the actual changes are monitored by a select few individuals. This model failed to gain enough motivation

from other contributors. On the other hand, the domain experts controlled how the story tests were generated and maintained in Chapter 6. Even though they were both using the same model, it seems to be more successful if the customers have the control of the story tests writing process.

In Chapter 6, we observed that people contributed most when they are given a set of loose structures to follow, but they were free to contribute in the format that is comfortable to them. Therefore, we believe that the initiation into such a knowledge building process should be low that almost anyone within the team (and even outside of the team) would be happy to contribute to the example repository of story tests. The success of Story Test Driven Development is not in producing better testing methods, but in fostering the community of contributors where everyone can produce rewards and motivations for each other.

CHAPTER 9: CONCLUSION

This dissertation investigated uses of Story Test Driven Development in Agile software development teams. There are three main research questions: 1) What problems are faced by Agile teams in practicing Story Test Driven Development? 2) Investigate the relationship between stories, teams and defects. 3) What are the factors that lead to successful adoption of Story Test Driven Development? We explored these questions using four case studies.

9.1 Summary of Findings

The findings led to four themes that are important in the success of Story Test Driven Development: Examples of the Domain, Story Test as Knowledge Repository, Rewards and Motivations, Community of Contributors. Our analyses suggest that Story Test Driven Development is a knowledge building process, rather than a software testing process.

Story Test Driven Development is a way for customers to engage in software product creation in a much more direct way than other methods. Because story tests are examples of the domain, these artefacts can be used not just for one version of software but used for building multiple versions of software or even multiple families of software. The value of the creating, collecting and maintaining story tests increases if the team is building a larger software product, especially if the product is a knowledge intensive type of software.

We also discovered that the main problem with the practice of Story Test Driven Development may lie in the difficulty with the customer participation. We presume that

customers require rewards and motivations that are different than developers. We have some evidence that the customers follow the selfish altruistic model. In order for this model to succeed, the team may require a community of contributors. The rewards and motivations may be directly linked to having a sizable community of contributors. In terms of the rewards, we discovered that the customers become both information consumer as well as the provider. Therefore, the rewards must be personal in nature rather than altruistic. Our studies indicate that the success of Story Test Driven Development may lie in fostering the community of contributors who are willing and able to create and maintain the story tests, but this research still requires more evidence.

9.2 Future Work

The studies in the dissertation suggests that the Story Test Driven Development is particularly useful in situation where the developers need to write software in customer's domain that is very unfamiliar and require a lot of training to understand. Our studies suggest that the story tests can be a useful communication medium for transferring domain knowledge, but it needs to be written in a format that is comfortable to the customers. In order to get the customers to participate in this process, the team needs to be building a community of contributors. It is not enough to just assign a task to one customer to provide these story tests. In addition, our studies suggest that the ownership of these tests must belong to the team, so that people are free and willing to contribute.

In the future work, we would need to look at the different types of customers and see how they react to writing story tests. It would be also interesting to look at the differences in roles and skills even within the customer group. It would be also

interesting to look at how different domains like to write their story tests and analyze how it influences the test automations for the developers as well as the overall adoption of Story Test Driven Development.

9.3 Main Contribution

The main contribution of this research is to approach Story Test Driven Development as a communication tool for conveying the domain knowledge using examples of the domain. Story Test Driven Development is a knowledge building process rather than a testing process. Story testing using examples is a way to spread out the domain knowledge and the necessary requirements information to the whole team using tests that can help confirm one's understanding of the domain as well as ensure that software behaves according to the examples provided. Instead of focusing on what is right for the test automation, we need to emphasize what is right for better communication with all stakeholders.

References

[A04] Andrea, J., Putting a Motor on the Canoo WebTest Acceptance Testing Framework, *Proc. of the 5th International Conference on Extreme Programming*, 2004, pp. 20-28

[A07] Andrea, J., Envisioning the Next Generation of Functional Testing Tools, IEEE Software, Vol. 24, Iss. 3, May/June 2007, pp. 58-662007

[A11] Agile Manifesto, www.agilemanifesto.org, retrieved April 2011

[A84] Axelrod, R., *The Evolution of Cooperation*, Basic Books, New York, 1984

[AA07] Agile Alliance Functional Testing Tools Visioning Workshop, Oct 2007, Portland, Oregon, www.agilealliance.org/show/1938

[AA08] Agile Alliance Functional Testing Tools Visioning Workshop, Agile 2008, Toronto, Canada

[AA11] AAFTT Community Mailing List, http://cf.groups.yahoo.com/group/aa-ftt/summary, retrieved April 2011

[AB04] Andersson, J., Bache, G., The Video Store Revisited Yet Again: Adventure in GUI Acceptance Testing, *XP 2004, LNCS 3092*, pp. 1-10

[AB05] von Ahn, L., Blum, M., Human Computation, Ph.D. Dissertation, Carnegie Mellon University, 2005

[ABL09] Abbattista, F., Bianchi, A., Lanubile, F., A Story-Test Driven Approach to the Migration of Legacy Systems, *XP* 2009, LNBIP 31, pp. 149-154

[ABS03] 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

[ABV05] Andersson, J., Bache, G., Verdoes, C., Multithreading and Web Applications: Further Adventures in Acceptance Testing, *XP 2005, LNCS 3556*, pp. 210-213

[AD06] Abrams, S., Deflorio, P., More than Videoconferencing: Trials of a Sidebar Voice System for Distributed Studies, 2nd Concurrent Engineering Workshop for Space Applications, European Space Agency, ESTEC; Noordwijk, The Netherlands, Oct 19-20, 2006

[AD08] von Ahn., L., Dabbish, L., Designing games with a purpose, Communications of the ACM, Vol 51, Iss.8, August 2008

[ALB06] von Ahn, L., Liu, R., Blum, M., Peekaboom: a game for locating objects in images, *Proc. Of the SIGCHI conference on Human Factors in Computing Systems*, April 2006

[An04] Andrea, J., Generative Acceptance Testing for Difficult-to-Test Software, *Proc. XP 2004*, LNCS Vol. 3092, pp. 29-37, 2004

[ARS07] 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

[B02] Beck, K., *Test Driven Development: By Example*, Addison-Wesley Professional, 2002

[B04] Beck, K., *Extreme Programming Explained: Embrace Change, Second Edition*, Addison-Wesley Professional, 2004

[B08] Bardram, J., Activity-based Computing Support for Agile and Global Software Development, *In Proc. Of 2008 Computer Supported Cooperating Work*, Workshop on Supporting Distributed Team Work, San Diego, CA, USA, 2008

[B90] Babbie, E., *Survey Research Methods,* Wadsworth, 1990

[B99] Beck, K., *Extreme Programming Explained,* Addison-Wesley, 1999

[Ba02] Barabasi, A., *Linked: How Everything is Connected to Everything Else.* Perseus Publishing, Cambridge, MA, 2002

[BB01] B.Boehm and V. Basili, Software Defect Reduction Top 10 List, *IEEE Computer,* Vol. 34, No. 1, pp. 2-6, January 2001

[Be02] Benkler, Y., Coase's Penguin, or Linux and The Nature of the Firm, *The Yale Law Journal,* 112:3, 2002, pp. 369-446

[BE05] Brandes, U., Erlebach, T., *Network Analysis: Methodology Foundations,* LNCS 3418, Springer, 2005

[BL01] Bergquist, M., Ljungberg, J., The Power of Gifts: Organizing Social Relationships in Open Source Communities, *Information Systems Journal* (11), 2001, pp. 305-320

[BSL99] Basili, V., Shull, F., Launibile, F., Building knowledge through families of experiments, *IEEE Transactions on Software Engineering,* vol. 25, pp. 456-473, 1999

[C04] Cockburn, A., *Crystal Clear: A Human-Powered Methodology for Small Teams,* Addison-Wesley, 2004

[C04] Cohn, M., *User Stories Applied: For Agile Software Development,* Addison-Wesley Professional, 2004

[C07] Crowdware, http://www.exampler.com/blog/2007/10/14/crowdware

[C09] Cohn, M., *Succeeding with Agile: Software Development Using Scrum,* Addison-Wesley Professional, 2009

[C74] Colton, T., *Statistics in Medicine*, Little, Brown, pp. 211, 1974

[CD07] Chubov, I., Droujkov, D., User Stories and Acceptance Tests as Negotiation Tools in Offshore Software Development, *XP 2007, LNCS 4536*, pp. 167-168

[CG09] Crispin, L., Gregory, J., Agile Testing: A Practical Guide for Testers and Agile Teams, Addison-Wesley Professional, 2009

[CH01] Crispin, L., House, T., Testing in the Fast Lane: Automating Acceptance Testing in an Extreme Programming Environment, XP/Universe Conference, 2001

[CHRP03] Cheng, L.-t.., Hupfer, S., Ross, S., Patterson, J., Jazzing up Eclipse wit collaborative tools, OOPSLA workshop on eclipse technology eXchange, Proc. of the 2003 OOPSLA workshop on eclipse technology eXchange, Anaheim, California, pp. 45-49, 2003

[CHS+03] Cheng, L.-t., de Souza, C. Hupfer, S., Patterson, J., Ross, S., Building collaboration into IDEs. *Queue*, Vol. 1, Iss. 9, pp. 40-50, 2003

[CHW01] 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

[CMK09] 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

[CKM09] 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

[CSGM06] Chen, J., Smith, M., Geras, A., Miller, J., Making Fit/FitNesse Appropriate for Biomedical Engineering Research, *XP 2006, LNCS 4044*, pp. 186-190

[D00] Dongen, S., Graph Analysis and Graph Clustering. In *Clustering by Flow Simulation*, Chapter 2, Wiskunde en Informatica Proefschriften, 2000, pp. 17-24

[D11] DSDM consortium, www.dsdm.org, retrieved April 2011

[D89] Dawkins, R., *The Selfish Gene*, 2nd ed. Oxford University Press, Oxford, 1989

[D92] Davis, A., Operational Prototyping: A New Development Approach, *Software*, 9(5), pp. 70-78, 1992

[DD08] Dybå, T., Dingsøyr, T., Empirical studies of agile software development: A systematic review, *Information and Software Technology*, 50 (2008), pp. 833-859

[DWM07] Deng, C., Wilson, P., Maurer, F., Fitclipse: A Fit-based Eclipse Plug-in for Executable Acceptance Test Driven Development, Proc. 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming (XP 2007)

[DL06] Derby, E., Larsen, D. Schwaber, K., *Agile Retrospectives: Making Good Teams Great*, Pragmatic Bookshelf, 2006

[DMG07] Duvall, P., Matyas, S., Glover, A., *Continuous Integration: Improving Software Quality and Reducing Risk*, Addison-Wesley Professional, 2007

[ER11] Energy Resources Conservation Board, http://www.ercb.ca/

[E96] European Software Institute, "European User Survey Analysis", Report USV_EUR 2.1, ESPITI Project, January 1996.

[EW06] Elssamadisy, A., West, D., Adopting agile practices: an incipient pattern language, *Proc. Of 2006 Confernece on Pattern Languages of Programs,* Portland, Oregon, USA

[F01] 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

[F07] Frost, R., Jazz and eclipse way of collaboration, *IEEE Software,* 24(6), 114-117, 2007

[F11] Fowler, M., XUnit, http://www.martinfowler.com/bliki/Xunit.html, retrieved April 2011

[F85] Fairley, R., Software Engineering Concepts, McGraw-Hill, New York, 1985

[F99] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., *Refactoring: Improving the Design of Existing Code,* Addison-Wesley Professional, 1999

[FF01] Fitzgerald, B., Feller, J., Open Source Software: Investigating the Software Engineering, Psychosocial and Economic Issues, *Information Systems Journal* (11), 2001, pp 273-276

[FK93] Kemerer, C., Fichman, R., Adoption of Software Engineering Process Innovations: The Case of Object Orientation, *Sloan Management Review/Winter,* 1993, pg. 7-22

[Fit11] Fit, fit.c2.com

[Fitn11] Fitnesse, fitnesse.org

[FitL11] FitLibrary, http://sourceforge.net/projects/fitlibrary/

[FitC11] FitClipse, http://sourceforge.net/projects/fitclipse/

[Fo11] Fowler, M., Specification by Example, http://www.martinfowler.com/bliki/SpecificationByExample.html, retrieved April 2011

[FP97] Fenton, N., Pfleeger, S., *Software Metrics: A Rigorous and Practical Approach*, PWS Publishing, 1997

[FPP98] Freedman, D., Pisani, R., Purves, R., *Statistics, 3rd Ed.*, Norton & Company, 1998

[G01] Greenhalgh, T., How to Read a Paper, 2nd ed. BMJ Publishing Group, London, 2001

[G11] Greenpepper, www.greenpeppersoftware.com

[GBGP07] 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

[GBL+04] 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

[GHHW05] 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

[GM07] Grewal, H., Maurer, F., Scaling Agile Methodologies for Developing a Production Accounting System for the Oil & Gas Industry, Proc. of Agile 2007 (2007)

[GMS05] Geras, A., Miller, J., Smith, M., Love, J., A Survey of Test Notations and Tools for Customer Testing, *XP2005*, LNCS 3556, pp. 109-117

[GS67] Glaser, B, Strauss, A., *Discovery of Grounded Theory: Strategies for Qualitative Research*, Chicago, Aldine, 1967

[H02] Highsmith, J., *Agile Software Development Ecosystems*, Addison-Wesley Professional, 2002

[H99] Hand, D., Statistics and Data Mining: Intersection Disciplines, ACM SIGKDD Exploration, Vol 1, Iss. 1, pp. 16-19, June 1999

[HF01] Hooks, I., Farry, K., Customer-centered products: Creating successful products through smart requirements management. American Management Association, New York, NY, 2001

[HH04] Hassan, A., Holt, R., The small world of software reverse engineering, *Working Conference on Reverse Engineering: IEEE Computer Society*, 2004

[HH08] Haugset, B., Hanssen, G., Automated Acceptance Testing: A Literature Review and an Industrial Case Study, Proc. of Agile 2008, pp. 27-38

[HH09] Hanssen, G., Haugset, B., Automated Acceptance Testing Using Fit, 42nd Hawaii International Conference on System Sciences, 2009, pp. 1-8

[HK06] Holmes, A., Kellogg, M., Automating Functional Tests using Selenium, *Agile Conference 2006*

[HMS59] Herzberg, F., Mausner, B., Snyderman, B., *The motivation to work*, New York, Wiley, 1959

[HO02] Hars, A., Ou, S., Working for Free? Motivations for Participating in Open Source Projects, *International Journal of Electronic Commerce*, (6:3), 2002, pp. 25-39

[HSB+08] Hall, T., Sharp, H., Beecham, S., Baddoo, N., Robinson, H., What do we know about developer motivation?, *IEEE Software*, 25(4), pp. 92-94

[I11] Rational Software Development Company, Rational Unified Process: Best Practices for Software Development Teams, Rational Software White Paper, TP026B, Rev 11/01, http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf, retrieved April 2011

[I90] IEEE, *IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Termionlogy*, IEEE Computer Society Press, Los Alamitos, CA, 1990

[J08] JUNG, http://jung.sourceforge.net/doc/index.html

[J11] JUnit.org, www.junit.org, retrieved April 2011

[J97] Jones, C. (eds), Software Quality: Analysis and Guidelines for Success, International Thomson Computer Press, 1997

[Ja11] IBM Rational Jazz, http://www-01.ibm.com/software/rational/jazz/, retrieved on May 2011

[Ji11] Jira, http://www.jira.com/

[JM01] Juristo, N., Moreno, A., *Basics of Software Engineering Experimentation*, Kluwer Academic Publishers, 2001

[JMD04] Javed, T., Maqsood, M., Durrani, Q., A Study to Investigate the Impact of Requirements Instability on Software Defects, ACM Software Engineering Notes, Volume 29, Number 4, May 2004

[JSG+06] Jacovi, M., Soroka, V., Gilboa-Freedman, G., Ur, S., Shahar, E., Marmasse, N., The Chasms of CSCW: A citation graph analysis of the CSCW conference, *Proc. of Computer Supported Cooperative Work*, Banff, Canada, pp. 289-298, Nov. 2006

[K03] 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

[K06] Kongsli, V., Towards Agile Security in Web Applications, Proc. of the Companion to the 21st ACM SIGPLAN Symposium OOPSLA, October 2006

[K07] Kongsli, V., Security Testing with Selenium, Proc. of Companion to the 22nd ACM SIGPLAN Conference on OOPSLA 2007, pp. 862-863

[K10] Keith, C., *Agile Game Development with Scrum*, Addison-Wesley Professional, 2010

[K11] Kerievsky, J. Storytesting, http://industrialxp.org/storytesting.html, retrieved April 2011

[K93] Karunanithi, N., A Neural Network Approach for Software Reliability Growth Modeling in the Presence of Code Churm, *Proc. of International Symposium on Software Reliability Engineering*, 1993, pp. 310-317

[K97] Kanigel, R., *The One Best Way: Frederick Winslow Taylor and the Enigma of Efficiency*, Penguin Books, 1997

[KAGNM96] Khoshgoftaar, T., Allen, E., Goel, N., Nandi, A., McMullan, J., Detection of Software Modules with High Debug Code Churn in a Very Large Legacy System, *Proc. of International Symposium on Software Reliability Engineering,* 1996, pp. 364-371

[KPGM09] Khandkar, S., Park, S., Ghanam, Y., Maurer, F., FitClipse: A Tool for Executable Acceptance Test Driven Development, Proc. of *XP 2009*, pp. 259-260

[KNR09] Kim, E., Na, J., Ryoo, S., Developing a Test Automation Framework for Agile Development and Testing, *XP* 2009, LNBIP 31, pp.8-12

[L00] Leffingwell, D., Widrig, D., *Managing Software Requirements: A Unified Approach,* Addison-Wesley, Reading, MA, 2000

[L11] Leffingwell, D., *Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise,* Addison-Wesley Professional, 2011

[M01] Miller, R., Collins, C. Acceptance Testing. Proc. XPUniverse 2001, July, 2001

[M03] Miller, R., Managing Software for Growth: Without Fear, Control and the Manufacturng Mindset, Addison-Wesley, 2003

[M05] Martin, R., The Test Bus Imperative: Architectures that Support Automated Acceptance Testing, *IEEE Software, July/August 2005,* pp. 65-67

[M07] Melnik, G., Empirical Analyses of Executable Acceptance Test Driven Development, University of Calgary, PhD Thesis, 2007

[M08] Mugridge, R., Managing Agile Project Requirements with Story Test Driven Development, IEEE Software, 25(1), 2008, pp. 68-75

[M11] Marick, B., Example-Driven Development, http://www.exampler.com

[M97] Montgomery, D., *Design and Analysis of Experiments, 4th edition,* John Wiley & Sons, 1997

[Ma08] Mathur, A., *Foundations of Software Testing,* Pearson Education, 2008

[MBN09] Martin, A., Biddle, R., Noble, J., XP Customer Practices: A Grounded Theory, Agile 2009, IEEE Computer Society, Chicago, 2009

[MBN09b] Martin, A., Biddle, R, Noble, J., The XP Customer Team: A Grounded Theory, Agile 2009, IEEE Computer Society, Chicago, 2009

[MC05] Mugridge, R., Cunningham, W., Agile Test Composition, *Proc. of the 6th International Conference on Extreme Programming,* 2005, LNCS 3556, pp. 137-144

[ME98] Munson, J., Elbaum, S., Code Churn: A Measure for Estimating the Impact of Code Change, *Proc. of IEEE International Conference on Software Maintenance,* 1998, pp. 24-31

[Mo03] Mogyorodi, G., What is Requirements-Based Testing?, CROSS TALK, The Journal of Defense Software Engineering, March 2003

[MLSM04] Muller, M., Link, J., Sand, R., Malpohl, G., Extreme Programming in Curriculum: Experiences from Academia and Industry, *XP* 2004, LNCS 3092, pp.294-302

[MM05] Melnik, G., Maurer, F., The Practice of Specifying Requirements using Executable Acceptance Tests in Computer Science Courses, Proc. 20th International Conference on Object-Oriented Programming, Systems, Languages (OOPSLA), ACM Press, 2005

[MM07] Melnik, G., Maurer, F., Multiple perspectives on Executable Acceptance Test-Driven Development, 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming (XP 2007)

[MM08] Martin, R., Melnik, G., Test and Requirements, Requirements and Tests: A Mobius Strip, IEEE Software, 25(1), pp. 54-59

[MMA00] Markus, M., Manville, B., Agres, C., What Makes a Virtual Organization Work?, *Sloan Management Review,* Fall 2000, pp. 13-26

[MMC06] Melnik, G., Maurer, F., Chiasson, M., Executable Acceptance Tests for Communicating Business Requirements: Customer Requirements, *In Proc. of Agile 2006 conference,* pp. 35-46

[MMR03] Mugridge, R., MacDonald, B., Roop, P., A Customer Test Generator for Web-based Systems, *XP 2003, LNCS 2675*, pp. 189-197

[MPS08] Moser, R., Pedrycz, W., Succi, G., A Comparative Analysis of the Efficiency of Change Metrics and Point Code Attributes for Defect Prediction, Proc. of 30th International Conference on Software Engineering, Leipzig, Germnay, pp. 181-190, 2008

[MRM04] Melnik, G., Read, K., Maurer, F., Suitability of FIT User Acceptance Tests for Specifying Functional Requirements: Developer Perspective, Proc. XP/Agile Universe 2004, LNCS, Vol. 3134, Springer Verlag, 60-72, 2004

[MR96] Maiden, N., Rugg, G., ACRE: Selecting Methods for Requirements Acquisition, *Software Engineering Journal*, 11(3), 183-192, 1996

[MS07] 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

[MT03] Mugridge, R., Tempero, E., Retrofitting an Acceptance Test Framework for Clarity, *Proc. Agile Development Conference*, 2003, pp. 92-98

[N11] NUnit.org, www.nunit.org, retrieved April 2011

[NB05] Nagappan, N., Ball, T., Use of relative code churn measures to predict system defect density, *Proc. of the 27th International Conference on Software Engineering*, St. Louis, USA, pp. 284-292, 2005

[NG04] Newman, M., Girvan, M., Finding and Evaluating Community Structure in Networks, *Physical Review, 69*, 026113, 2004

[NM05] Nielsen, J., McMunn, D., The Agile Journey Adopting XP in a Large Financial Services Organization, *XP 2005, LNCS 3556*, pp. 28-37

[O78] Ohno, T., *Toyota Production System: Beyond Large-Scale Production*, Productivity Press, 1978, translated into English in 1988

[OM08] Ordelt, H., Maurer, F., Acceptance Test Refactoring, *XP2008*, Limerick, Ireland, Springer, 10-14 June 2008

[OP09] 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

[OWB04] Ostrand, T.J., Weyuker, E.J., Bell, R.M., Where the Bugs Are, *Proc. of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis* (ISSTA), pp. 86-96, 2004

[P10] Pichler, R., *Agile Product Management with Scrum: Creating Products that Customers Love*, Addison-Wesley Professiona, 2010

[P63] Popper, K. R. (1963) Conjectures and Refutations: The Growth of Scientific Knowledge, New York: Basic Books

[P94] Pfleeger, Experimental Design and Analysis in Software Engineering, *ACM Sigsoft, Software Engineering Notes*, 19(4), 20(1), 20(2), 20(3), 1994-1995

[Pa02] Patton, M., Qualitative Research and Evaluation Methods, Sage Publications, 2002

[PF02] Palmer, S., Felsing, J., *A Practical Guide to Feature-Driven Development*, Prentice-Hall, 2002

[PK08] Port, D., Korte, M., Comparative Studies of the Model Evaluation Criterions MRRE and PRED in Software Cost Estimation Research, Proc. of International Symposium on Empirical Software Engineering and Measurement (ESEM), Oct. 9-10, Kaiserlautern, Germany, 2008

[PM08] Park, S., Maurer, F., The Benefits and Challenges of Executable Acceptance Testing, Workshop on Scrutinizing Agile, In Conjunction with ICSE 2008

[PM09] Park, S., Maurer, F., Communicating Domain Knowledge in Executable Acceptance Test Driven Development, *XP2009, LNBIP 31,* May 2009, pp. 23-32

[PP03] Poppendieck, M., Poppendieck, T., *Lean Software Development: An Agile Toolkit*, Addison-Wesley, 2003

[PP06] Poppendieck, M., Poppendieck, T., *Implementing Lean Software Development From Concept to Cash*, Addison-Wesley Professional, 2006

[PP09] Poppendieck, M., Poppendieck, T., *Leading Lean Software Development: Results are Not the Point*, Addison-Wesley Professional, 2009

[PW03] Pawson, R., Wade, V., Agile Development Using Naked Objects, *Proc. of the 4th XP 2003, LNCS 2675*, 97-103

[R04] 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

[R11] Robot Framework, http://code.google.com/p/robotframework/

[R93] Robson, C., *Real World Research: A Resource for Social Scientists and Practitioners-Researchers,* Blackwell, 1993

[R03] Reichlmayr, T., The agile approach in an undergraduate software engineering course project, *33rd Annual Frontiers in Education, 2003,* pp. 13-18, S2C Vol. 3

[R04] 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

[RW73] Rittel, H., Webber, M. "Dilemmas in a General Theory of Planning", *Policy Sciences*, 4, 155-169, 1973

[RMM05] Read, K., Melnik, G., Maurer, F., Student Experiences with Executable Acceptance Testing, Proc. Agile 2005 Conference, 2005

[RMM05b] Read, K., Melnik, G., Maurer, F., Examining usage patterns of the FIT acceptance testing framework, Proc. 6th International Conference on eXtreme Programming and Agile Processes in Software Engineering, (XP 2005), Lecture Notes in Computer Science, Springer Verlag, 2005

[RPA08] Read, D., Properjohn, G., Going Agile- A Case Study, *19th Australian Software Engineering Conference, 2008,* Perth, Australia

[RPT+08] 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, International Conference on Software Engineering 2008, Leipzig, Germany, pp. 361-370

[RTD+08] 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

[RTCT07] 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

[S01] Schwaber, K., Beedle, M., Agile Software Development with Scrum, Prentice Hall, 2001

[S02] Shaw, M., What Makes Good Research in Software Engineering, International Journal of Software Tools for Technology Transfer, 2002, Vol. 4, No. 1, pp. 1-7

[S03] Steinberg, D., Using Instructor Written Acceptance Tests Using the Fit Framework, *LNCS*, Vol. 2675, Springer-Verlag, pp. 378-385, 2003

[S04] Schwaber, K., *Agile Project Management with Scrum,* Microsoft Press, 2004

[S07] Schwaber, K., *The Enterprise and Scrum,* Microsoft Press, 2007

[Sh07] Sharp, H., The role of physical artefacts in agile software development team collaboration, *2nd International Workshop on Physicality*, Lancaster University, UK, 2007

[S09] Schwaber, K., The Sprint Review: Mastering the Art of Feedback", www.scrumalliance.org/articles/124-the-sprint-review-mastering-the-art-of-feedback, retrieved April 2011

[S10] Schmidt, J., Lyle, D., *Lean Integration: An Integration Factory Approach to Business Agility,* Addison-Wesley Professional, 2010

[S66] Sabidusi, G., The centrality index of a graph, *Psychometrika,* Vol. 31, pp. 581-603, 1966

[S87] Strauss,A., *Qualitative Analysis for Social Scientists.* Cambridge, England: Cambridge University Press, 1987

[SC98] Strauss, A., Corbin, J., *Basics of Qualitative Research: Technqieus and Procedures for Developing Grounded Theory,* 2nd ed. Thousand Oaks, Sage, 1998

[SC88] Siegel, S., Castellan, J., *Nonparametric Statistics for the Behavioral Sciences,* 2nd Ed. McGraw-Hill International Editions, 1988

[SK07] Shuja, A., Krebs, J., *IBM Rational Unified Process Reference and Certification Guide: Solution Designer,* IBM Press, 2007

[SN08] Sauve, J., Neto, O., Teaching Software Development with ATDD and EasyAccept, SIGCSE 2008, March 12-15, Portland, Oregon, USA, pp. 542-546, 2008

[SNC05] Sauve, J., Neto, O., Cirne W., EasyAccept: A Tool to Easily Create, Run and Drive Development with Automated Acceptance Tests, AST' 05, pp.111-117

[SP04] 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

[SP07] Sailer, K., Penn, A., The Performance of Space – Exploring Social Spatial Phenomena of Interaction Patterns in an Organization, *Architecture and Phenomenology Conference*, May 13-17, 2007, Haifa, Israel

[SRP07] Sharp, H., Rogers, Y., Preece, J., *Interaction Design: Beyond Human-Computer Interaction,* Wiley, 2007

[SRSF06] Sharp, H., Robinson, H., Segal, J., Furniss, D., The Role of Story Cards and the Wall in XP teams: A Distributed Cognition Perspective, Agile 2006, Minneapolis, MN, USA, pp. 65-75

[SS97] Sommerville, I., Sawyer, P., *Requirements Engineering: A Good Practice Guide,* John Wiley & Sons, Chichester, England, 1997

[SSO05] Schwarz, C., Skytteren, S., Ovstetun, T., Aut-AT: An Eclipse Plugin for Automatic Acceptance Testing of Web Applications, *OOPSLA* 2005, pp. 182-183

[St09] Stolberg, S., Enabling Agile Testing through Continuous Integration, Agile 2009

[St95] Stake, *The Art of Case Study Research,* SAGE Publications, 1995

[S07] Sumrell, M., From Waterfall to Agile – How does a QA Team Transition?, Agile 2007

[T11] Taylor, F., *The Principles of Scientific Management,* Harper & Brothers, New York, 1911

[T79] Trivers, R., The Evolution of Reciprocal Altruism, *Quaterly Review of Biology*, 46, 1971, pp. 35-57

[TD09] Talby, D., Dubinsky, Y., Government of an Agile Software Project, Proc. of the 2009 ICSE Workshop on Software Development Governance, 2009, pp. 40-45

[TKHD06] Talby, D., Keren, A., Hazzan, O., Dubinsky, Y., Agile Software Testing in a Large-Scale Project, *IEEE Software, July/August* 2006, pp. 30-37

[T98] Torvald, L., What Motivates Free Software Developers?, *First Monday,* (3:3), 1998, http://firstmonday.org/issues/issue3_3/torvalds/index.html

[WRH+00] Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., and Wesslen, A., *Experimentation in Software Engineering – An Introduction,* Kluwer Academic Publishers, 2000

[W03] Wiegers, K., *Software Requirements, Second Edition,* Microsoft Press, 2003

[W04] Wurowski, J., *The Wisdom of Crowd,* Random House, 2004

[W71] Winer, B.J., *Statistical Principles in Experimental Design*, McGraw-Hill, pg. 14, 1971

[Wa04] Watts, D., Six Degrees: The Science of a Connected Age, W.W. Norton & Co., 2004

[WJR90] Womack, J., Jones, D., Roos, D., *The Machine that Changed the World,* Rawson Associates, 1990

[WL04] Watt, R., Leigh-Fellows, D., Acceptance Test Driven Planning, *LNCS, Vol. 3134,* Springer-Verlag, pp. 43-49, 2004

[V11] Videosurf, www.videosurf.com

[Y03] Yin, R., Case Study Research: Design and Metrics, Sage Publications, 2003

[Y94] Yin, R., *Case Study Research: Deign and Methods (Second Edition),* Sage Publications, Thousand Oaks, CA, 1994

[YRG09] Yague, A., Rodriguez, P. Garbajosa, J., Optimizing Agile Processes by Early Identification of Hidden Requirements, *XP 2009, LNBIP 31*, pp. 180-185

[ZW98] Zelkowitz, M., Wallace, D., Experimental Models for Validating Technology, *IEEE Computer*, 31(5), pp.23-31, 1998

[ZN02] Zowghi, D., Nurmuliani, N., A study of the impact of requirements volatility on software project performance, In Proc of the 9th Asia-Pacific Software engineering Conference, Washington, DC, USA, *IEEE Computer Society*, 2002

[ZN04] Zimmermann, T., Nagappan, N., Predicting defects using network analysis on dependency graphs, Proc. of 30th International Conference on Software Engineering, Leipzig, Germany, pp. 531-540, 2008

Appendix I: Ethics Approval

The following pages are intentionally left blank in accordance with the Faculty of Graduate Studies regulations.

APPENDIX II: COPYRIGHT RELEASE FORM

The following pages are intetionally left blank in accordance with the Faculty of Graduate Studies regulations.

APPENDIX III: INTERVIEW QUESTIONS

Interview Guide

Date/Time:

Purpose of the Study:

This research aims to determine how software requirements are specified in executable specifications, discover the tool adoption process and the benefits and problems that people encounter with the existing executable acceptance testing tools.

What Will I Be Asked To Do?

You will be asked about your experience with executable acceptance testing tool in the current or previous projects that you worked on. You will be asked about your role in the project, how the team used the tool and why the tool was chosen. The interview will take about 30 minutes. The interview will be audio taped. Your participation in this research is voluntary. If at any time you feel uncomfortable, you may withdraw from the study – the audio recordings will be discarded and any data gathered from your participation removed. Knowledge of your participation in the project will remain anonymous.

Questions

1. Tell me about the project that you were involved that used Fit.
    a. How did you get started on the project?
    b. What is your role in the team?
    c. At what point of the project did you get involved?
2. Tell me about the processes involved in introducing Executable acceptance Test Driven Development to your team
    a. Who introduced executable acceptance testing in your team?
    b. How did you get the other team members involved?
    c. What were the obstacles in introducing the tool to your team?
    d. Why did you choose to practice executable acceptance test driven development?
    e. How did you overcome the adoption problems?
3. Who writes the specifications?
4. Tell me about how the tool works.s
    a. Why did you choose this tool?
    b. What kind of problems did you try to solve using Fit?
    c. What were the problems/shortcomings with the tool?
5. Did you notice benefits to using Fit?