Understanding the Efficacy of Test Driven Development

Ling (Angela) Li

A dissertation submitted to

Auckland University of Technology

in partial fulfillment of the requirements for the degree of

Master of Computer and Information Sciences

2009

School of Computing and Mathematical Sciences

Primary Supervisor: Jim Buchan

Secondary Supervisor: Anne Philpott

Table of Content

List of Figures

List of Tables

List of Abbreviations

TDD	Test Driven Development
XP	Extreme Programming
OO	Object-Oriented
BA	Business Analyst
LOC	Lines of Code

Attestation of Authorship

Ling (Angela) Li

Acknowledgements

This dissertation was written at the School of Computing and Mathematical Sciences at the Auckland University of Technology.

I would like to take this opportunity to thank all those people who contributed to and made it possible for me to complete this dissertation. First and foremost I would like to thank my supervisor, Jim Buchan for all his knowledge, efforts and guidance through the entire study. I would like to express my gratitude to him for his encouragement and valuable contribution to this research. I also wish to thank my second supervisor, Anne Philpott, for her ideas, suggestions. Special thanks are extended to all of the participants of this research for their assistance and valuable time, since without them the research would not have been possible.

To all my lecturers in the MCIS course, I would like to thank them for the guidance and imparting the knowledge and skills I required to complete this dissertation.

I want to thank my families and friends for all their support, encouragement, love and prayers.

This research was approved by the Auckland University of Technology Ethics Committee on 27 April 2009, AUTEC Reference number 09/46.

Abstract

Test Driven Development (TDD) is a software development practice in which unit and acceptance test cases are incrementally written prior to writing the production code and guiding the design of the target software. Recently, TDD has gained considerable attention among practitioners and researchers, outside its original context in Extreme Programming (XP), with the promise of a number of benefits to the software development process. Proponents of TDD claim that TDD enhances code quality, application quality and developer productivity compared with a traditional Test-Last development methodology. There are some published empirical studies of TDD that support these claims, but few of these are in an industrial setting and generally focus on measurements of quality or productivity. This study adds to this evidence-based research on TDD by studying practitioners' perceptions of the benefits of TDD. Furthermore, this study seeks to go further than previous studies in this area and gain a deeper understanding of the efficacy of TDD by investigating the underlying factors contributing to the benefits of using TDD. This is important to inform future tool support and process improvements of TDD. In addition it may suggest strategies for increasing the adoption level of TDD.

This research takes a single case study approach. TDD practitioners' perceptions of the benefits of TDD and the contributing factors were collected in five semi-structured interviews conducted in a medium-sized software company located in Auckland, New Zealand. The results of the interviews were analyzed using a thematic analysis technique looking for patterns and themes that aligned with the three general benefits identified from literature, namely improvements to code quality, application quality and developer productivity.

This study contributes empirical evidence where experienced TDD practitioners clearly perceive that using TDD provides a number of benefits and that these align closely with those claimed by proponents of TDD and evidence in related empirical studies. The main underlying factors contributing to these benefits identified in this research can be summarised as: increased confidence in refactoring and modifying code, broader test coverage, deeper understanding of requirements, enhanced code comprehensibility, reduced error scope and severity, and improved developer satisfaction. The benefits of TDD are

perceived as strongly linked to developers' capabilities in refactoring, and writing high quality tests.

The findings of this research add to the existing empirical body of knowledge on the use of TDD in an industrial context, as well as strengthening the case for using TDD in this type of software development environment. The research also indicates that current integrated development environments need to improve the level of support they provide for TDD activities.

Chapter 1 - Introduction

1.1 Background and Motivation

Test Driven Development (TDD) is not, despite its name, a testing technique but rather a software development methodology derived from Extreme Programming (XP) and the principles of the Agile Manifesto (Agile Alliance, 2000). Taking Janzen and Saiedian's definition of a development methodology (2005), TDD defines the order, control, and evaluation of the basic tasks involved in creating software. In traditional development, tests are built after the target product feature exists, also known as "Test-Last". In contrast to the traditional Test-Last development methodology, TDD requires developers writing automated tests prior to developing functional code in small, rapid iterations. This is highlighted in the rule suggested by Beck (2003) for developers using TDD: "Don't write even a single line of code if an automated test has failed".

A number of authors note that with the rise of use of XP and Agile practices, TDD has gained added visibility. Both proponents and researchers of TDD claim that the use of TDD yields several benefits, including better code quality, enhanced application quality and developer productivity. For example Beck (2001), an early adopter and proposer of TDD, notes that, in his experience, TDD simplifies the design by producing loosely coupled and highly cohesive code structure. This has been further supported by Jazen and Saiedian (2008), who conducted three controlled experiments and a case study in this area, providing both quantitative and qualitative evidence that TDD produces simpler, more loosely coupled and highly cohesive code compared to Test-Last development. Both Crispin's case study (2006) and Muller and Hagner's controlled experiment (2002) claim that their research results showed that TDD increases the degree of customer satisfaction of the final application. Other researchers such as George and Williams (2003), and Williams, Maximilien et al. (2003) provide significant supporting evidence that TDD improves the application quality by reducing defect rates. Moreover, the results from some experiments conducted by TDD researchers Erdogmus, Morisio et al. (2005), Janzen and Saiedian (2006) and Guptand and Jalote (2007) indicate that TDD improves the overall productivity by reducing the design and rework efforts.

While these empirical studies of TDD provide some evidence that the claimed benefits are realized in practice, few of these are set in an industrial context. This study therefore seeks to contribute to the evidence-based research in this area by conducting the investigation in an industrial context. In addition previous studies have largely ignored investigating the factors that contribute to these advantages and generally focus on quantitative measurements of, for example, quality or productivity benefits. In order to fill this gap, this research aims to gain a deeper understanding of the benefits by exploring the contributing factors. Furthermore, rather than focusing on investigating the benefits by considering quantitative measures of those benefits, this study focuses on understanding practitioners' perceptions of these benefits and the contributing factors. Researchers can claim benefits, but it is whether or not practitioners perceive those benefits as "real" that will influence the pressures for adoption of TDD as well as tool support and process improvement. This, in turn, will influence the research agendas of TDD researchers.

This study is also expected to provide insights that will be useful for improving current software development practice and possibly inform tool design. It should be of value to both the software development research community and practitioners.

The next section formalizes the motivation for this research into some explicit research objectives.

1.2 Research Objectives

The aim of this research is to obtain a deeper understanding of the efficacy of TDD by investigating the underlying contributing factors that make TDD an effective software development practice. The primary focus of this research is to investigate the factors from current practitioners' perceptions with their experience of using TDD in commercial projects. So the main objectives of this research are:

To investigate the benefits of using TDD in practice by analyzing practitioners' perceptions.

To explore the factors that contribute to the benefits of TDD in practice based on practitioners' perceptions.

Additionally, there is frequently a gap between literature and practice (Nikula, Sajaniemi et al. 2000), identifying the difference as well as investigating the reason would add values to the research area. Therefore another objective of this research is:

To compare and contrast practitioners' perceptions of the benefits of TDD in practice with the findings from literature.

Moreover, this study also expected to grant some recommendations to improve current software development practice. So the other objective of this research is:

To investigate practitioners' satisfaction with using TDD and any suggested changes.

The next section describes the research questions to be answered to achieve the stated research objectives.

1.3 Research Questions

In order to meet the research objectives stated above, the research questions have been formed to investigate the issues embodied in the research objectives as:

1. *a. What are the benefits of TDD realized in practice?*
   b. What are the disadvantages of using TDD in practice?
   c. Do the benefits of TDD realized in practice, align with those claimed in literature
2. *What are the underlying factors that contribute to these benefits?*
3. *Are practitioners satisfied with their current TDD practices and level of training?*

Table 1.1 displays the relationship between the research objectives and research questions.

Research Objectives	**Research Questions**
To investigate the benefits of using TDD in practice by analyzing practitioners' perceptions.	Q 1a. What are the benefits of TDD realized in practice? Q 1b. What are the disadvantages of using

	TDD in practice?
To compare and contrast practitioners' perceptions of the benefits of TDD in practice with the findings from literature.	Q 1c. Do the benefits of TDD realized in practice align with those claimed in literature?
To explore the factors that contribute to the benefits of TDD in practice based on practitioners' perceptions.	Q 2. What are the underlying factors that contribute to these benefits?
To investigate practitioners' satisfaction with using TDD and any suggested changes.	Q 3. Are practitioners satisfied with their current TDD practices and level of training?

Table 1.1-Link between the research objectives and research questions

1.4 Research Scope

This research intends to investigate the TDD practice through current practitioners' perspectives in New Zealand. In order to do so, this research is focused on the users' perceptions from the people in the organizations whose main business is software development and who have adopted TDD for developing software. In terms of geographical area, the scope is Auckland region, New Zealand. The reason for this is: easy access by the researcher.

1.5 Dissertation Outlines

This research contains five chapters:

- **Chapter 1 – Introduction:** introduces the background of the topic and research rationale for conducting this research.
- **Chapter 2 – Literature review:** provides a review of past works that are relevant to this research on efficacy of Test Driven Development.
- **Chapter 3 – Research methodology:** discusses and justifies the research methodology and the research methods employed in conducting this research.

- **Chapter 4 – Research findings and discussion:** presents a summary of the findings, analysis of the findings, as well as the explicitly answering the research questions.
- **Chapter 5 – Conclusion:** presents the conclusion of the research, limitations of this research and the further research opportunities.

1.6 Conclusion

This chapter provides the reader with the background and rationale of this research as well as its objectives. The research objectives provide a basis for formulating the research questions. This chapter also covers the research scope and an outline of the dissertation structure is presented.

Chapter 2 – Literature Review

2.1 Introduction

This chapter considers the state of current research as it relates to the objectives of this dissertation. It summarises related theory and practices from the research literature and provides a wider context and motivation for this study. Section 2.2 explores the key concepts of TDD and section 2.3 providing insights into the suggested TDD benefits by reviewing empirical studies in this area. The last section of this chapter (section 2.4), summarises the findings from literature, as well as identifying the gaps in the literature. A model which structures the high level TDD benefits suggested from literature is presented in this chapter. This model is subsequently used as the framework for data gathering and analysis in this research.

2.2 The Key Characteristics of TDD

TDD, as introduced in Chapter1, is a technique for developing software that uses the writing of tests to guide the design of the target software. There are 4 key characteristics of TDD suggested in literature (see for example Janzen and Saiedian 2005; Erdogmus, Morisio et al. 2005): small unit focus, test-orientation, frequent regression testing and automated testing. These characteristics have been used to structure this discussion of TDD research literature.

- **Small Unit Focus**

Developers' focus on detailed features in development is a specific characteristic of TDD. In the context of XP, features are essentially high-level requirements that the customer identifies and prioritizes, and the term "user story" was adopted in XP to represent an end-user feature that has a number of pieces of functionality or tasks. TDD is described as developing software in very short iterations with little upfront design. The working process of TDD is: developers start with one user story, then design and implement a piece of functionality. The functionality increases at a relatively consistent rate. As noted by Williams, Maximilien et al. (2003), new functionality is not considered properly implemented unless the new unit test cases and every other unit test cases written for the code base runs properly.

Take the example of a user story that has two different aspects of functionality, A and B. By comparing the two working processes illustrated in Figure 2.1, in traditional Test-Last development process, a developer firstly writes the production code for both functionalities to implement the whole story, then writes tests to validate the whole story, and finally reworks the production code until all tests pass. But in TDD, a developer firstly writes tests for specifying and validating the functionality A and then produces the code to make the tests pass. This will be reworked until functionality A is fully completed. The work cycle is repeated for the implementation of functionality B. The testing of functionality A is repeated everytime functionality B is refined and tested. The whole story is completed at the end of the second cycle. Therefore, one difference between TDD and the tradtional Test-Last development approach is that developers are focusing on a small unit for design and implementation when using TDD.

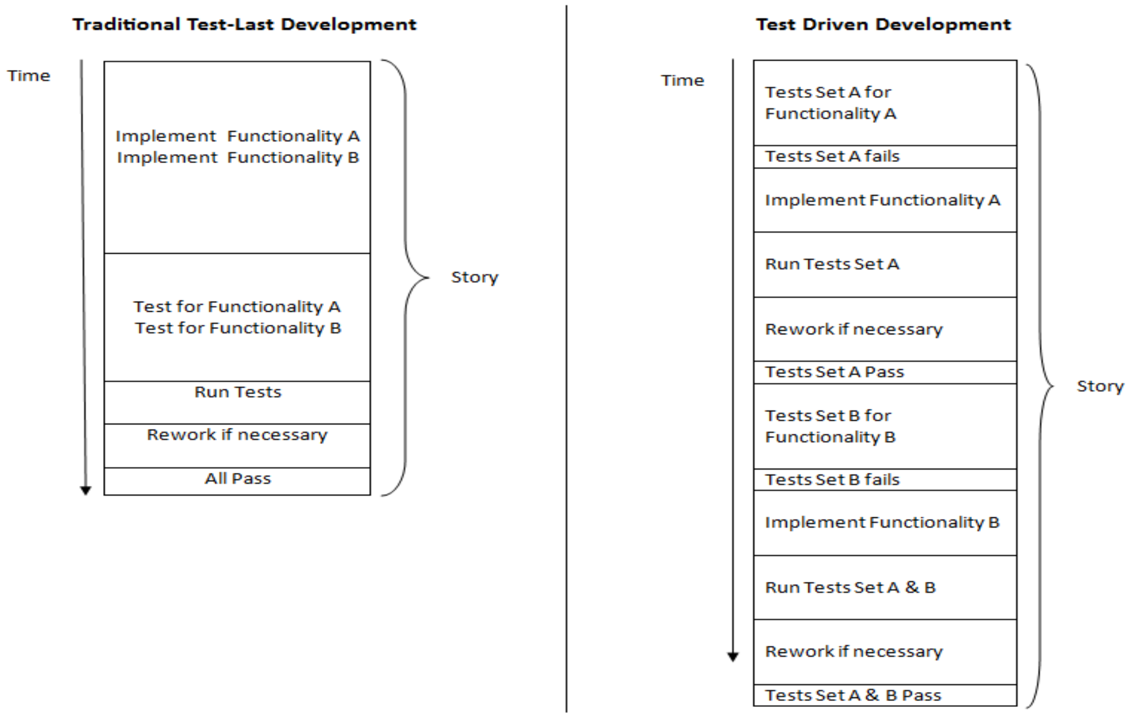


Figure 2.1-Working Process of Traditional Test-Last Development vs. TDD (Adapted from Erdogmus, Morisio et al. 2005)

- **Test-orientation**

TDD is also known as test-first programming, so tests drive the coding activity. Erdogmus, Morisio et al. (2005) refer to this as “test-orientation”. It is one of the most significant characteristics of TDD. In traditional Test-Last development process, tests are generally built after the target product feature exists and are for verification and validation purposes. Whereas, in TDD, developers write tests that generally break the system (called breakpoint tests) for each small piece of functionality, before starting coding the functionality. As Lui and Chan (2007) emphasise, these tests are used as the specification for the functionality in addition to verification and validation.

Moreover, as highlighted in the definition of TDD given by the Agile Alliance, developers should “produce exactly as much code as will enable that test to pass” (as quoted in Janzen and Saiedian, 2005). Therefore, tests in TDD also are used to define the scope of production code under development, since code that is not needed to pass the test is out of scope.

In summary, there are three special features of the TDD approach represented within this characteristic: making breakpoint tests, using tests to specify functionality and scope; and using tests to “drive” the production of code.

- **Frequent Regression Testing**

Formulated from the concept and principles of TDD, the development process comprises the following steps (Liu and Chan 2007):

1. Add a unit test for one functionality in the selected user story
2. Run all tests including the new added one, and see the new test is failed as the code has not yet been implemented
3. Write code to pass the new test
4. Run all tests and see them all succeed
5. Refactor the code if necessary
6. Run all tests and see them all succeed
7. Repeat step 1-6 for another functionality

This shows that regression testing is an integral part of TDD. All of the test cases will be run before and after new code is added to the code base and the process is iterated when each piece of functionality is implemented. If the regression test fails, even if the new functional test passed, this means that the new functionality has introduced a defect that affects the previously written code. Therefore, frequent regression testing is one of key characteristics in TDD.

- **Automated Tests**

Another key characteristic of TDD is that tests in TDD are automated. There is an important rule defined by Beck (2003) said that: "Have all tests run at 100% all the time". Therefore, tests in TDD need to be able to be executed continuously over the course of the development process. Consequently, automated tests are indispensable in TDD. Also sets of automated tests are useful for regression testing.

There are automated unit testing frameworks developed to simplify both the creation and execution of software unit tests. Janzen and Saiedian (2005) suggest that they have played an important role in the emergence of TDD as they were argued to aid to improve and increase developer productivity. Erich Gamma and Kent Beck developed JUnit, an automated unit testing framework for Java, essential for implementing TDD with Java. JUnit-like frameworks have been implemented for several different languages, creating a family of frameworks referred to as xUnit.

2.3 Suggested TDD Benefits

A number of benefits are claimed and evidenced in literature regarding the use of TDD in software development, as compared to a traditional Test-Last approach. These can be categorized as the following high level benefits: improved code quality, improved application quality, and enhanced developer productivity. These categories are depicted graphically in Figure 2.2 and this framework is used to structure the discussion of the benefits of TDD from literature for the rest of this chapter, and again in the discussion of the results from the field study. The following sub sections discuss these high level benefits, their meanings and any underlying factors from literature. In Chapter 4 these findings from literature are then compared with the findings from the field study to confirm (or otherwise) and extend the notions of the benefits of using TDD.

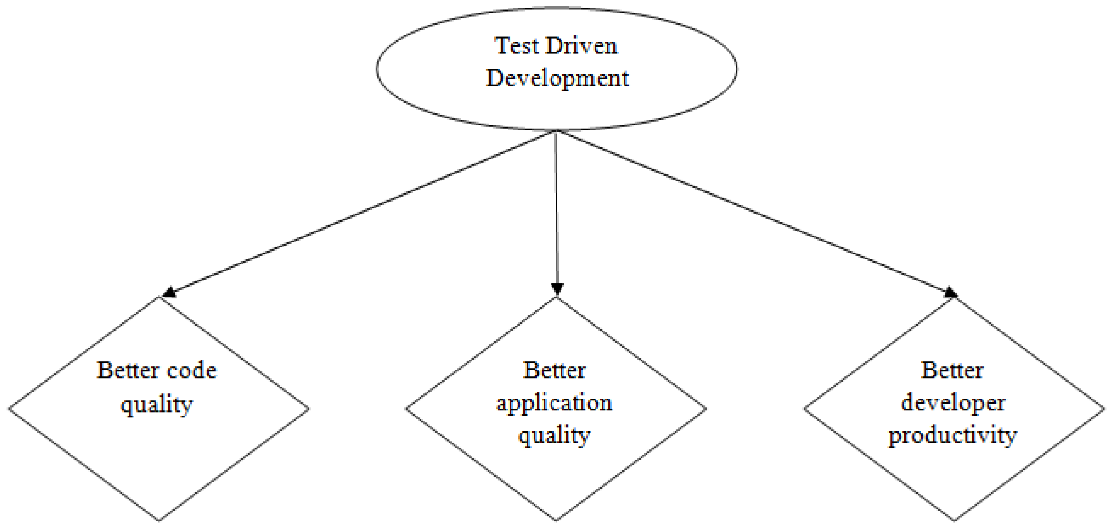


Figure 2.2-High-level TDD Benefits

2.3.1 Code Quality

There are claims stated in literature that TDD improves the code quality by providing simpler design and cleaner code. For example, Beck (2001) asserted that TDD simplifies the design because “Test-first code tends to be more cohesive and less coupled than code in which testing isn’t part of the intimate coding cycle”. The research by Janzen and Saiedian (2008) also demonstrated that TDD is better in producing simple, loosely coupled and highly cohesive code. The research conducted by Janzen and Saiedian (2008) included three quasi-controlled experiments and one case study involving 12 undergraduate and graduate software engineering students at the University of Kansas to compare the TDD approach with the traditional Test-Last approach. The experiments interwove the approaches and mixed up their order in completing individual projects. The first quasi-experiment involved a Test-Last project with no automated tests, followed by a second phase of the same project completed with a TDD approach. The second quasi-experiment involved a Test-Last project followed by a TDD project. The third quasi-experiment involved a TDD project followed by a Test-Last project. The case study interviewed the developers from 15 software projects completed in one development group over five years. The 15 projects included the five TDD and Test-Last projects from the industry quasiexperiments, and 10 used a test last approach. The results showed that the TDD had smaller modules and fewer methods per class than Test-Last; TDD had lower complexity and better cohesion.

Massol and Husted (2003), in their well-known book on JUnit testing, claim that the use of TDD is effective at producing cleaner code and also that it “eliminates duplication of code”. No specific empirical evidence is presented in the book to support this, however. Crispin (2006) does provide some supporting evidence for cleaner code with TDD from her own experience with using TDD in software development projects. She attributes this to TDD encouraging the development of more clearly defined scope of features or stories. So, by having to writing code that just passes the associated test and no more, the goal and scope of the required code is smaller and clearer and so the developer is encouraged to stay within this limited scope and write less code. Crispin also provides evidence from her case studies that code written using TDD is easier to understand.

It is worth noting that, the aspects of better code quality are contributing factors to improved application quality and developer productivity. Improved code quality has been included as a principal high-level benefit of TDD because it is mentioned so frequently in literature in the context of TDD.

2.3.2 Application Quality

Advocates of TDD claim that it enhances the quality of the software application developed using TDD. Common aspects of application quality mentioned in literature are reliability and correctness. Spinellis (2006) defines software reliability as its capability to maintain the specified level of performance under the specified conditions (i.e. fewer defects). Godbole (2004) describes application correctness as the degree to which the application software meets specified requirements and user expectations. Both industrial and academic studies provided significant evidence to support the occurrence of these benefits in practice.

The research results from Crispin (2006), Muller and Hagner (2002) both indicated that TDD leads to the better requirement understanding and this consequently improves the correctness of the software. Crispin (2006) investigated TDD benefits by investigating TDD practitioner’s experiences and perceptions and found that practitioners felt they could understand the requirement better with TDD compared to the traditional Test-Last development practice. The practitioners also acknowledged that customers were more satisfied with the software developed with TDD because the developers felt they anticipated the features better. Muller and Hagner (2002) studied the effectiveness of TDD programming by conducting a controlled experiment with two groups of subjects in an academic setting. Both groups developed a small program, one using a TDD approach and the other group using the traditional Test-Last approach. By comparing the results from the two groups, the authors observed that the TDD approach appeared to support better program understanding and the program met the user specifications to a higher degree.

George and Williams (2003) also conducted a controlled experiment but with practitioners. They showed that TDD produces higher quality software with fewer bugs. The experiment used 24 professional programmers to develop the same small Java program (size 200 LOC). Programmers were divided evenly into two groups with one using TDD approach and the other using the traditional Test-Last approach. The authors found that the TDD group produced higher quality code which passed 18% more functional tests. They further stated that such findings may result from TDD encouraging more frequent and tighter verification. They also indicated that the programmers following a Test-Last process often did not write the required tests after completing their code, which might cause inadequate testing and reduce the application reliability. TDD, on the other hand, has the potential of increasing the level of testing, and this may contribute to higher quality software.

Two industrial case studies also provide strong evidence that TDD helps with producing more reliable software. It is shown in both those studies that the defect rate is lower in the project developed using TDD compared to the project developed traditionally. In one study Williams, Maximilien et al. (2003) carried out a case study in IBM which reported up to a 40% reduction in defect density for the TDD group compared to the other group. In a more recent study of two similar case studies conducted with slightly different context factors conducted at Microsoft, Bhat and Nagappan (2006), observed that a similar result that the use of TDD reduces the number of defects. Their results of the two case studies also indicated that more test coverage contributed to lower defect density.

2.3.3 Developer Productivity

Advocates of TDD claim that its use enhances the overall productivity of a development team by reducing total project development effort - defined as time spent directly on the project including analysis, design, code, test, fix and review. Although TDD has received criticism for increasing the coding time, advocates claim TDD enhances the *overall* productivity by reducing the efforts on design, testing and fixing. A number of studies can be found in the literature to support these claims. Three recent examples of these studies are now discussed.

Erdogmus, Morisio et al. (2005) conducted an experiment to evaluate the use of TDD against the Test-Last strategy with two groups of undergraduate students. They were given the task of developing a small program using alternate strategies. Both groups followed incremental development and applied unit testing. The difference was that the TDD group wrote functional tests before coding and did just enough coding to pass the tests, whereas the Test-Last group first coded functionality and then wrote functional tests. The researchers found that the TDD group wrote more tests but were more productive in terms of overall development effort, specifically less rework effort at the testing and fixing stages.

The research conducted by Janzen and Saiedian (2006) was also a controlled experiment with undergraduate students in a software engineering course. Students in three groups completed semester-long programming projects using either an iterative TDD, iterative Test-Last, or linear Test-Last approach. Results from this study indicated that TDD programmers were more productive than the Test-Last programmers since the TDD team spent less effort per line-of-code and 57% less effort per feature than the Test-Last team.

Moreover, research results gathered by Gupta and Jalote (2007) provide further evidence in support of the notion. Similar to the two experiments stated above (Erdogmus, Morisio et al. 2005; Janzen and Saiedian 2006), the research is also a controlled experiment with students participants. Twenty-two computer science students were divided into two groups to implement a small application using TDD and the traditional Test-Last approach. The results indicated that the TDD approach resulted in considerable saving of upfront-design efforts.

These three empirical studies are carried out using students in an academic setting and as discussed by Carver, Jaccheri et al. (2003), they provide a useful mechanism for researchers to try out their ideas before conducting the research in an industrial setting. Context factors such as the project size and complexity are generally quite different in a commercial project. It follows then that the results from a study in an educational context may not simply transfer over to the use of TDD in a commercial project in an industrial context. As mentioned in Chapter 1, part of the motivation for this study is to add to the body of empirical studies set in an industrial context.

It should be noted that a number of researchers have provided evidence from empirical studies that the use of TDD increases the time for the coding phase (see for example: George and Williams 2003; Canfora, Cimitile et al. 2006; Bhat and Nagappan 2006). George and Williams (2003) also indicated that transitioning to the TDD mindset is difficult, and it might be one factor that increases the efforts put in the coding phase with the use of TDD. These studies, however, do not include end-to-end productivity in their investigations and make no claim about *overall* productivity. The previously discussed studies argue that productivity gains in other development stages (e.g. design, testing and fixing) more than compensates for increased coding time.

2.4 Conclusion

A number of studies have been conducted to investigate the efficacy of TDD. They provide evidence that supports the occurance of the benefits of using TDD in industrial and academic settings. From this literature review, the claimed benefits of using TDD are conceptualised into three high level categories. The meanings for each category of high level benefit are also identified.

Most of the existing studies describe and verify the benefits, but few of these probe into the reasons or contributing factors for these benefits. This dissertation seeks to confirm these benefits but also investigate the factors contributing to the realisation of the benefits.

Table 2.1, Table 2.2 and Table 2.3 summarise the benefits of using TDD from literature. In Chapter 4, they are used for comparing the findings from the field study with findings from literature, and are extended with new findings from the field study.

High Level Benefits	Better code quality
Meanings	• Simple Design (Janzen and Saiedian, 2008) • Clean Code (Crispin, 2006) • No duplication of code (Massol and Husted, 2003) • Low coupling of code modules (Beck,2001; Janzen and Saiedian, 2008) • High cohesiveness of code modules (Beck,2001; Janzen and Saiedian, 2008) • Easier to understand-more meaningful (Crispin, 2006)
Contributing Factors	• Developers have clearer knowledge of the scope of stories or features (Crispin, 2006; Janzen and Saiedian, 2005)

Table 2.1-Benefits related to code quality claimed in literature

High Level Benefits	Better application quality
Meanings	• Simple Design (Janzen and Saiedian, 2008) • Low defect rates (Bhat and Nagappan, 2006) • High customer satisfaction with application (Muller and Hagner, 2002) • Passes more functional tests (George and Williams, 2003)
Contributing Factors	• Better understanding of requirements. (Crispin, 2006) • TDD may encourage more frequent and tighter verification and validation (George and Williams, 2003) • TDD has potential of increasing the level of testing (George and Williams, 2003) • More tests coverage (Bhat and Nagappan, 2006)

Table 2.2-Benefits related to application quality claimed in literature

High Level Benefits	Better developer productivity
Meanings	• Less effort to produce final product from start to finish (Erdogmus, Morisio et al., 2005; Janzen and Saiedian, 2006)
Contributing Factors	• Less rework effort at the testing and fixing stages (Erdogmus, Morisio et al., 2005) • Savings in up-front design effort (Gupta and Jalote,2007)

Table 2.3-Benefits related to developer productivity claimed in literature

Before representing the findings and analysis, it is important to understand how these findings were obtained by describing and justifying the research methodology and the data collection and analysis methods. These are therefore the subject of the next chapter.

Chapter 3 – Research Methodology

3.1 Introduction

This chapter describes and justifies the methodology for this research in detail. The selection of the research methodology and specific data gathering and analysis methods is driven by the research aims. The discussion and justification of the chosen methodology for this research are stated in section 3.2. The data collection method and processes are presented in section 3.3. Section 3.4 explains the data analysis method and procedures. Finally, section 3.5 provides the conclusion of this chapter.

3.2 Research Methodology

As stated in Chapter1, the intention of this research is to gain insights into the phenomenon of TDD's effectiveness through practitioners' perceptions. Since the research focuses on practitioners' perceptions rather than, for example, direct observation, an interpretive research approach is appropriate for this research. An interpretivist epistemology is common and well accepted in Information Systems research in this type of study which involves the investigation of people's interaction within their social and cultural context (Klein and Myers, 1999).

A qualitative approach to data gathering is adopted. According to Myers (1997), qualitative research is focused on social and cultural phenomena and deals with collecting and analyzing qualitative data, whereas quantitative research focuses on natural phenomena and deals more with quantitative data. The data to be gathered and analyzed in this study is the practitioners' perceptions from their experiences of using test driven development, therefore a qualitative research approach is taken.

This study aims to gain insights from practitioners who have experiences with using TDD in commercial projects, so a case study is chosen as the research method. According to Klein and Myers (1999), a case study offers the opportunity for studying the phenomenon in its commercial context in the real world. This is supported by several other studies in this area. For example both Bhat and Nagappan (2006), and Maximilien and

Williams (2003) use a case study approach to investigate the benefits of using TDD within realistic industrial settings.

Case studies can be designed to deal with a single case or multiple cases. According to Yin (2003) "by focusing on single case, a researcher is generally able to obtain deeper understanding of the phenomenon under investigation compared to focusing on multiple cases, so single case may be used to confirm or challenge a theory". This aligns well with the stated objective of this research to obtain a deep understanding of the efficacy of TDD through practitioners' perceptions; therefore a single-case study is selected for this study.

With the selection of single case study as the research method, the study phases are discussed in detail in the following section.

3.3 Data Collection

3.3.1 Data Collection Method

The focus of this research on practitioners' perceptions is based on a desire to "know" the practitioners more closely and understand their experiences in this area. Valenzuela and Shrivastava (2007) argue that a suitable data collection method for getting the story behind a participant's experiences is an interview. Moreover, Yin (2003) points out that data collection using interviews can have the advantage of focusing directly on the topic of interest by asking both open as well as targeted questions related directly to the topic. Adler and Adler (1998) point out that interviews also provide opportunities for the researcher to enhance the validity of the results obtained by using participants' quotes to support conclusions. With these points in mind, the use of an interview for data collection aligns well with the aims of this research.

There are three commonly recognized types of interviews: (1) structured, (2) unstructured, and (3) semi-structured. According to Berg (1998), a structured interview uses a formally structured schedule of interview questions. In structured interviews, interviewers ask all respondents the same series of pre-established questions with a limited set of response categories. It is assumed that the questions "scheduled" in the interview instruments are sufficiently comprehensive to solicit from subjects all (or nearly all) information relevant to the study's topic(s). The current study, however, is exploratory in nature and all the relevant information is not known beforehand. If a structured interview approach were taken, then certain information about the phenomenon, which might be useful, could be missed. Therefore, a structured interview is not suitable for this research.

On the other hand, unstructured interviews do not utilize schedules of questions. In unstructured interviews, interviewers develop, adapt and generate questions and follow-up probes appropriate to the given situation (Berg 1998). So there would be higher possibility that the interview may go out of context. Employing unstructured interviews could limit the strength of focusing on certain aspects of the phenomenon that the researcher is interested in.

Semi-structured interviews locate somewhere between the extremes of completely structured and completely unstructured. This type of interview involves the implementation of a number of predetermined questions and/or topics of interest (Berg 1998). Therefore, it gives interviewees the freedom to express and explain their perceptions by answering the open-ended questions. Consequently interviewers may obtain rich information about the phenomenon. In addition the interactive nature of the semi-structured interview encourages the development of a rapport and trust between the interviewee and interviewer. This is desirable if interviewees are to feel comfortable sharing experiences and challenges. Moreover, it allows the interviewer to explore the phenomenon of interest in a deeper manner than a structured approach by asking additional or follow-up questions (Collis and Hussey, 2003). Therefore, considering the desire to explore the perceptions of TDD in some depth and obtain rich information, semi-structured appears to be the most suitable interview type for this research. Thus, semi-structured interview is selected as the data gathering method in this research.

3.3.2 Interview Instrument

The design and development of the interview instrument (interview question guide) is based on the research objectives and research questions stated in Chapter 1, the framework of analysis (Figure 2.2) as well as findings from the related literature (Table 2.1, Table 2.2 and Table 2.3) developed in Chapter 2. The interview guide can be found in Appendix A.

The interview guide is structured in three parts regarding the interview questions. In the first part, context information about the participants, projects and the organization are collected. This includes users' experiences, project size and the development techniques/tools. This provides a context for this study. The second part of the interview guide contains questions directly related to the participants' perceptions of the benefits and disadvantages of using TDD in a recent development project. Following good practice, this section is structured with open-ended questions at the start, followed by more directed questions related to the specific benefits of interest. This included probing questions to uncover the contributing factors of the benefits and disadvantages identified. In the final section of the interview guide, participants' overall satisfaction with using TDD in the project, as well as their suggestions for process improvements are sought.

Table 3.1 shows how the interview questions relate to the research questions, Table 3.2 displays how the interview questions relate to the research objectives, Table 3.3 lists how the interview questions relate to the context factors, and Table 3.4 maps the interview questions to the dimensions of the framework of analysis.

Interview Questions	**Research Questions**
9 – 10	1a. What are the benefits of TDD realized in practice?
12 – 13	1b. What are the disadvantages of using TDD in practice?
9.1 – 9.7	1c. Do the benefits of TDD realized in practice align with those claimed in literature
11, 14	2. What are the underlying factors that contribute to these benefits?
15 – 19	3. Are practitioners satisfied with their current TDD practices and level of training?

Table 3.1-Relationship of interview questions and research questions

Interview Questions	**Research Objectives**

9 – 10 , 12 – 13	To investigate the benefits of using TDD in practice by analyzing practitioners' perceptions.
11,14	To explore the factors that contribute to the benefits of TDD in practice based on practitioners' perceptions.
9.1 – 9.7	To compare and contrast practitioners' perceptions of the benefits of TDD in practice with the findings from literature.
15 – 19	To investigate practitioners' satisfaction and suggestions with using TDD.

Table 3.2-Relationship of interview questions and research objectives

Interview Questions	**Context Factor**
1 - 5, 6.4	User Experience
6.1, 6.3	Project Size
6.2, 7	Development Techniques

Table 3.3-Relationship of interview questions and context factors

Interview Questions	**Framework of Analysis Dimensions**
9.1, 10 – 14	Code Quality
9.2-9.3, 10 – 14	Application Quality
9.4-9.7, 10 – 14	Developer Productivity

Table 3.4-Relationship of interview questions and framework of analysis

3.3.3 Participant Selection

Based on the aim and the scope of this research, the criteria used for inviting organizations to participate in the single case study are:

1. A software development company, which has utilized TDD in software development projects to a significant degree.
2. Auckland based, ensuring the researcher can have face-to-face interviews in order to obtain richer data, as well as avoiding the need for travelling.

The criteria used for recruiting the participants from the selected organization are:

1. Experienced senior software professionals since they are likely to have a good knowledge of the software development lifecycle.
2. Professionals who have experience with both TDD as well as the traditional Test-Last approach, in order to ensure participants can compare insights on these two development methodologies.

In accordance with the above criteria, the research was conducted in a medium sized product-driven software company based in Auckland, New Zealand. The company has a product developed using a Test-Last approach for 15 years, followed by over 2 years of a Test Driven approach more recently. This company was identified from the researchers' existing contacts and agreed to participate when invited. This was the only organization to be invited.

The software manager of the partnering company provided a list of five individuals, who had senior roles in the project being studied and were likely to meet the required criteria. These candidate participants were provided with a Research Information Sheet and a Participant Consent Form (see Appendices B and C) and invited to participate in the study. All five agreed to participate and were subsequently interviewed. There is a possibility of bias in this selection process, since the final selection of the individuals and how well they met the selection criteria, was at the discretion of their software manager.

3.3.4 Interview Protocols

Interviews were held at participants' usual place of work. Each participant was interviewed on a one-to-one basis for about half an hour in a private meeting room. After brief introductions and a summary of the research, participants were given the opportunity to clarify any aspects of the Participant Consent Form and asked to sign it if they hadn't already. The format of the interview was then explained to the interviewee so they knew what to expect. The consent to record the interview was then reconfirmed with interviewees and the recording started.

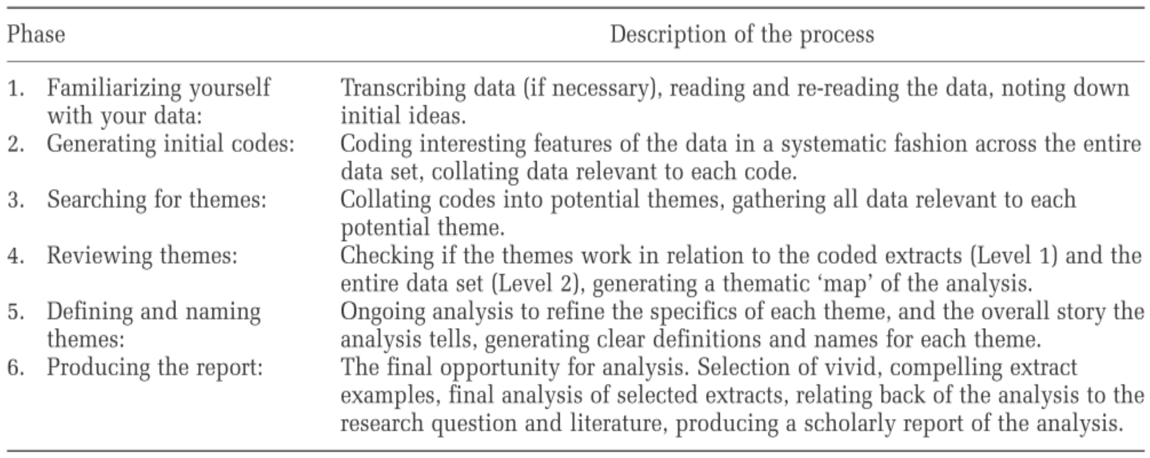
There were no follow-up interviews and no opportunity to bring the interviewees together for further discussion as a group.

With the use of semi-structured interview techniques, interviews started off with open questions to allow the participants to express their views with little prompting from the interviewer. If necessary, more directed questions or probing questions were followed up in order to make sure the research questions were addressed adequately.

The data were gathered in the form of detailed interview notes as well as a recording of the interview. The capturing of the interview notes was simplified using pre-prepared information capture forms with the same structure as the interview question guide. It was not planned to transcribe the interview recordings in their entirety, but for them to act as a backup to clarify any uncertainty in the interview notes and provide supporting quotes.

3.4 Data Analysis

The data collected from the individual interviewees were analyzed at the project level using a thematic analysis technique. This technique of coding the collected data into themes is commonly used for analyzing qualitative interview data (see for example Boyatzis 1998). As described by Owen (1984) in his seminal paper on this topic, themes were identified according to the criteria of recurrence, repetition and forcefulness. Recurrence refers to the situation where several parts of the interview data have the same meaning even though the actual words may be different. Repetition is the situation where key phrases or words are repeated several times throughout the interview data. The criterion of “forcefulness” relates to some signal from the interviewee that the sentence or idea is strongly emphasized. This may be signaled through body language, change in voice volume or some emphasizing phrase (e.g. “it all boils down to...”). An inductive approach to identifying themes from the interview data has been adopted, as described by Braun and Clarke (2006). The goal was to identify themes regarding the benefits, disadvantages and contributing factors without trying to fit them in to preconceived idea, although as noted by Braun and Clarke (2006), the “researchers cannot free themselves of their theoretical and epistemological commitments”. The analysis of the themes followed the 6 steps suggested in Braun and Clarke (2006) and was summarised in Figure 3.1. Part of the thematic analysis, related to research question 1c, involved a comparison of the identified themes associated with the benefits of TDD with the claims and findings from literature in this area.



Phase	Description of the process
1. Familiarizing yourself with your data:	Transcribing data (if necessary), reading and re-reading the data, noting down initial ideas.
2. Generating initial codes:	Coding interesting features of the data in a systematic fashion across the entire data set, collating data relevant to each code.
3. Searching for themes:	Collating codes into potential themes, gathering all data relevant to each potential theme.
4. Reviewing themes:	Checking if the themes work in relation to the coded extracts (Level 1) and the entire data set (Level 2), generating a thematic 'map' of the analysis.
5. Defining and naming themes:	Ongoing analysis to refine the specifics of each theme, and the overall story the analysis tells, generating clear definitions and names for each theme.
6. Producing the report:	The final opportunity for analysis. Selection of vivid, compelling extract examples, final analysis of selected extracts, relating back of the analysis to the research question and literature, producing a scholarly report of the analysis.

Figure 3.1-Themes Analysis Steps (Braun and Clarke, 2006)

3.5 Conclusion

This chapter justifies the selection of the single case study as a suitable research methodology. Furthermore the use of semi-structured interviews for data collection and thematic analysis for data analysis has been described and rationalised. The selection of interview participants and the interview protocol used have also been detailed. The next chapter presents the findings and results of this research, as well as discussions and links with the research questions.

Chapter 4 - Findings and Discussion

4.1 Introduction

The previous chapter explains and justifies the research methodology and outlines the steps the research followed. The research findings are presented and discussed in this chapter.

This chapter firstly describes the case project and the backgrounds of the participants and shows that these align well with the desired criteria (section 4.2). This is followed by a description, analysis and discussion of the research findings (section 4.3). This section presents the findings of the research and discusses the findings around the high level benefits represented in the framework of analysis (Figure 2.2) in detail. These findings are compared and contrasted with findings in literature, as well as analysed for a deeper understanding of the contributing factors of the claimed benefits of TDD. Finally, section 4.4 concludes the chapter by drawing together the main results.

4.2 Summary of the Case

Based on the answers to the interview questions 1 to 8 (see Appendix A), this section presents the background of the participants, as well as the case project being investigated.

4.2.1 Participants' Backgrounds

Interview questions 1 to 5 (see Appendix A) relate to information about the participants' backgrounds. The answers to those questions indicate that their software development experience is in the range of 7-15 years. Moreover, they all have recent experiences of using TDD as well as a traditional Test-Last approach in software development. None of the participants reported having had any formal training in TDD. The roles of the interviewees include the software engineer, team leader and business analyst (BA), which helped to obtain views of TDD practices from a number of perspectives. This may provide a balanced set of views. When asked to describe TDD, the participants' conceptualizations of TDD were accurate and consistent with the descriptions of TDD found in literature.

The backgrounds of the interviewees were a good fit to our criteria for participation and this was confirmed in the insightful observations they made about their perceptions of TDD practice.

4.2.2 Project Background

Answers to interview questions 6 to 8 (see Appendix A) provide the information on the TDD project this study investigates. Table 4.1 summarises this context information. The company in which the study was conducted has spent 15 years developing a product using a Test-Last approach (referred to as the “legacy project” from here on). Those involved reported some difficulties in maintaining this legacy project code; and, in order to correct this and avoid the same issue in the future, management made the decision to rebuild the product with an Agile software development approach. A new project manager was appointed who was experienced in Agile development methodologies. This manager promoted TDD as the approach because he had seen the benefits from his previous experiences. TDD was subsequently adopted for rebuilding the product over the last 3 years (referred to as “TDD project” from here on). Developers were sent to work on the legacy project and the TDD project alternately. Therefore, most developers in the company have experiences with both development approaches.

Team Experience	Mix of Junior, Intermediate and Senior
Team Size	Range from 12 to 35 people
Project Size	Large, rewrite a 15 years old product, over two years
Development Techniques	TDD, also adopted Pair Programming
Development Tools	Java, JUnit, Eclipse

Table 4.1-TDD Project Information

4.3 Findings and Discussion

In this section the interview data are analyzed and the themes identified are categorized with the dimensions of the framework of analysis (Figure 2.2). The findings of the interview data indicate that the benefits of using TDD participants perceived align well with the three high level benefits claimed in literature. Moreover, the perceptions of the meanings of those high level benefits are similar to the findings in literature. Analysis of the interview data uncovers several new factors that contribute to the benefits of using TDD. . Tables 4.2, 4.3 and 4.4 highlight the findings related to the three high level benefits defined in the framework of analysis. Each table also includes the benefits of using TDD and the contributing factors identified in literature (previously summarised in Tables 2.1, 2.2 and 2.3). They provide a clear comparison of the findings in this study and the findings from the literature.

High Level Benefits	Better code quality	
	Findings in this research	**Findings from literature**
Meanings	• Simple Design • Low coupling • Clean Code • Meaningful Code	• Simple Design (Janzen and Saiedian, 2008) • Clean Code (Crispin, 2006) • No duplication of code (Massol and Husted, 2003) • Low coupling of code modules (Beck,2001; Janzen and Saiedian, 2008) • High cohesiveness of code modules (Beck,2001; Janzen and Saiedian, 2008) • Easier to understand-more meaningful (Crispin, 2006)
Contributing Factors	• More confident/ willing to change/refactor • Forced to consider (analysis + code) small chunks of functionality • Early analysis of	• Developers have clearer knowledge of the scope of stories or features (Crispin, 2006, Janzen and Saiedian, 2005)

	requirements , results in deep understanding • Just make code for test and no more • Meaning embedded in test • Fewer comments required	

Table 4.2-Findings of benefits related to code quality in this research vs. findings from literature

High Level Benefits	Better application quality	
	Findings in this research	**Findings from literature**
Meanings	• High customer satisfaction with application • Low defect rates	• Simple Design (Janzen and Saiedian, 2008) • Low defect rates (Bhat and Nagappan, 2006) • High customer satisfaction with application (Muller and Hagner, 2002) • Passes more functional tests (George and Williams, 2003)
Contributing Factors	• Come back to BA frequently to clarify the requirement and gain better requirement understanding • Better coverage of boundary cases • Instant feedback when	• Better understanding of requirements. (Crispin, 2006) • TDD may encourage more frequent and tighter verification and validation (George and Williams, 2003)

	break something and fixes have been done immediately • Increased test density and code are well tested	• TDD has potential of increasing the level of testing (George and Williams, 2003) • More tests coverage (Bhat and Nagappan, 2006)

Table 4.3-Findings of benefits related to application quality in this research vs. findings from literature

High Level Benefits	Better developer productivity	
	Findings in this research	**Findings from literature**
Meanings	• Less effort to produce final product from start to finish	• Less effort to produce final product from start to finish (Erdogmus, Morisio et al., 2005; Janzen and Saiedian, 2006)
Contributing Factors	• Less rework required • Meaning is clearer so others can rework more easily and after a long period of time. • More in touch with progress – test passing or failing • Developer satisfaction is improved with: - early frequent success with passing the tests - more confident to make changes to the existing application - more confident with	• Less rework effort at the testing and fixing stages (Erdogmus, Morisio et al., 2005) • Savings in up-front design effort (Gupta and Jalote,2007)

	the code they build	

Table 4.4- Findings of benefits related to developer productivity in this research vs. findings from literature

The findings related to each claimed high level benefit are discussed in detail in the following sub sections. Each sub section discusses the themes identified in participants' perceptions related to the benefit as well as the contributing factors they perceived for each benefit.

4.3.1 Code Quality

Similar to the claims in literature, participants have perceived that using TDD improves the code quality. They indicated that the use of TDD helps with producing simple designs, with low coupling and clean and meaningful code. They contrasted this with the traditional Test-Last approach.

Participants noted that good developers always aim to produce code with a simple design, no matter with which type of development methodology. They perceived TDD as a useful practice to encourage this discipline and integrate it into everyday practice, making it "front of mind". The following quote from one interviewee illustrates this theme:

> *"TDD helps developers towards the simple design, keeps the things typically OO structure, pushes developers towards separated components..."*

Participants identified several factors which they considered as contributing strongly to this benefit.

Interviewees perceived that, compared to a Test-Last approach, it was beneficial to develop a deeper understanding of the functionality required *prior* to writing production code, and TDD "forced" this by having to write tests first. They described this as resulting in them having a clearer idea about the objects and the methods they were going to need in the production code and made it easier for them to keep the design simple.

Another major benefit of using TDD identified by the participants is the inherent development of a set of automated tests. Developers described the instant feedback they could get about any changes they make by running all the automated tests, as providing a much enhanced level of confidence to make those changes. They were more willing to refactor to improve design or try out new ideas, rather than the more "if it isn't broke, don't touch it" feeling they previously had with Test-Last practices. Moreover, developers described the "focus on smaller pieces of functionality" (e.g. one piece of functionality in a user story) that TDD encourages, as beneficial. They perceived that making changes against a small chunk of code is easier than making changes after the whole story finishes. The participants described themselves as more willing to make changes to improve the design of the code during the implementation process.

Interviewees also indicated that the code developed in the TDD project seemed to be "cleaner", with less extraneous or superfluous code. Participants identified that the main reason for this benefit is that writing the tests first makes the scope of the production code clearer to them. They then tended to produce just enough code to pass the test**,** "not too much, not too little". This contributing factor is also identified in literature by Crispin (2006).

Participants also perceived that code is generally more "readable" in the TDD project, which they saw as contributing to the code quality and reducing time for rework. The participants attributed this to writing tests first that specify what the production code will do. They tended to want to use meaningful names for test names as well as variables, classes in the test. This meant that it was easy to recognize what production code it referred to and what the test did. This use of meaningful names naturally carried over to the production code resulting in code that required fewer comments and was easier to understand both from the code and its associated test(s). Participants also indicated that, because the code is more meaningful, developers found it easier to comprehend and rework code written by others or re-visited by them a long time later. Participants described meaningful test cases as replacing some of the extra documentation for describing the function they needed in Test-Last development. Compared with the legacy project, participants reported fewer comments in the code in the TDD project. The following quotes highlight these themes:

> *"In TDD, developers were required to write meaningful tests, which can be served as the documentation for describe the functionality. Thus, the documentation is embedded in the code."*

“There is no need to have a large bit of documentation outside of the code or inside it to describe the functionality itself, as the test class would give a good idea of what the relevant class should be doing.”

In summary, participants perceived that using TDD results in simpler designs. They claimed that the initial design tended to be simpler with low coupling because TDD encourages small incremental development and a deeper understanding of functionality before implementing the code. They also noted that they were more willing to refactor the code to improve the design *during* development because TDD provides quick feedback on the effects of the refactoring.

Furthermore, participants perceived the production code as “cleaner” with less unnecessary code and fewer comments needed. This was attributed to the focus on producing code that passes the specified test(s) and no more, with a clear scope. They observed that much of the information and semantics that were traditionally captured in comments became embodied in well written tests. They also perceived that well written tests help developers to understand the functionality easier.

4.3.2 Application Quality

The data collected from the research indicate that most interviewees have strong perceptions that TDD enhances the application quality with an improved degree of client satisfaction and application reliability. These findings are similar to the claims found in the literature.

Similar to the findings obtained by Crispin (2006), participants also perceived that TDD helps developers to get better requirement understanding. Several participants reported that producing the tests first with meaningful names to specify the functionality, makes developers take more time, early in the development lifecycle, to analyse the scenarios that need to be handled. They saw this as helping them to achieve better requirement understanding and consequently improving the correctness of the software and the level of customer satisfaction with the application. Two interviewees stated that, compared to the legacy project, developers contacted the client or the Business Analyst (BA) (the client proxy in the company) more often to discuss the requirements in the TDD project. The following quote typifies this perception:

"Certainly, when you write a test makes you question, you have to think about certain scenarios a little bit more, that makes you question your functionality that you developing the code for, and then the questioning make you understand the requirement better."

Interviewees also indicated that they found using TDD improves software reliability with fewer defects. According to the answers given by interviewees, developers were found to think more about the boundary cases needing to be covered by having to write breakpoint tests first. The work steps of TDD is to create the tests for one functionality first and run all the tests to see the new tests fail as the code had yet to be implemented, then write the code to make the new added tests pass. Interviewees reported that, compared to writing production code directly to implement the functionality, they found developers taking more time to think about what cases needed to be covered when they need to write the tests which should fail at first. Since boundary cases were more carefully covered at the implementation stage, fewer defects were found at the end of the development cycle. A typical example described by participant is: "previously (working with legacy project) you found it will be more small defects in the test phase… but under TDD, you won't actually get that a lot small defects, also probably even get less large ones".

Another contributing factor they identified as contributing to software quality was that developers are able to identify and fix problems instantly during the implementation process, which corroborates one factor claimed by George and Williams (2003). Participants reported that by running all automated tests before and after new functionality is implemented, they very quickly knew whether the new functionality would pass its test and also whether the new functionality had "broken" any other part of the code. This allowed immediate action to be taken (rework of the problem or test) and avoided compounding problems or inappropriate tests:

"If that (new functionality) breaks the old test cases, you go back immediately to evaluate if they need to be changed or evaluate if they still need to be satisfied."

A further factor contributing to this benefit was noted by participants: when the problem is revealed immediately, the context is still fresh in the developer's mind, and it is much easier for the developer to fix it then, rather than fix it months later.

Another factor which the participants perceived as important to improving application reliability is the increase in testing density that following TDD produced. Participants provided insight, similar to the findings claimed by Bhat and Nagappan (2006), that the team which produced more tests had fewer defects in their code. TDD encouraged better test coverage of the code. Code wasn't written unless it had a test to pass. George and Williams (2003) also make this claim. Participants claimed that with Test-Last development there was a tendency to leave testing out if time was short - or to only test parts that were considered critical. They noted that it was quite rare to go back and write a test for code that had no test but had been running well in production. Participants indicated that with the use of TDD they are more confident to deliver the application because they are sure the code is well tested. This confidence added to their overall satisfaction and motivation for coding, compared to a Test-Last approach.

In summary, participants strongly perceived a benefit of TDD as the improvement of application quality, in terms of reliability and degree of client satisfaction. They credited this to achieving a better requirements understanding using TDD because writing the tests first promotes earlier analysis of requirements often resulting in early clarification with the client. They also perceived that TDD encourages developers to produce better coverage of boundary cases, resulting in fewer application defects. Another factor that the participants perceived as contributing to application reliability is the continuous testing and feedback TDD provides, so errors tend to be picked up early and frequently before they become critical. In addition, they noticed that TDD increases the test density, which improves the test coverage for the application.

4.3.3 Developer Productivity

In terms of improved developer productivity, the results of the study show that participants had similar insights to those stated in the literature. Participants stated quite strongly that using TDD generally improves the overall development productivity. They explained that use of TDD may increase the time spent on initial coding (tests and production code) compared to a Test-Last approach, but that TDD reduces the effort required during the maintenance stage, as well as often reducing the amount of rework required. Interestingly, the participants' didn't explicitly identify the benefit claimed by

Gupta and Jalote (2007), namely that of less effort spent in the design of the production code.

In line with others' findings indicating that using TDD generally increases the coding time (e.g. George and Williams, 2003), interviewees stated that with using TDD "developers didn't implement the task faster". They attributed this in some part to the steep learning curve and the overhead of learning a new method. The following quote from an interviewee exemplifies this idea:

> *"It's a bit hard to get your head around... people are used* to write test *afterwards..."*

Participants also mentioned that they perceived learning how to write good tests and "what tests to write" as an overhead that increased the overall coding effort. Obviously this could be the case with non-TDD good practice, but the use of TDD "forced" the participants into making more effort in this area.

Participants indicated quite strongly that another factor contributing to the extra effort in coding using TDD is the "the lack of IDE support". Most development tools are able to create the classes and objects easily for developers, but such IDE support is weak in the test driven mode. In particular, participants noted the lack of support of their Eclipse-based development environment in creating tests for non-existent classes or methods, as needed by TDD, was problematic.

However, although the coding time is considered to be longer in TDD because of the factors mentioned above, participants noted that this often improved as experience with TDD grew.

Similar to the findings in literature (e.g. Janzen and Saiedian, 2006), participants viewed TDD as improving developer productivity in terms of the *end-to-end* development cycle, from requirements to implementation. They credited one of the contributing factors to this as a reduction in the amount of rework required with the use of TDD. TDD requires running all automated tests continuously, so problems can be identified and fixed quickly at the coding stage which participants saw as requiring less effort than fixing the problems later on. They explained the reason for this as follows. When a problem is revealed immediately following a short burst of testing and implementation activity, the problem context is still fresh in the programmer's mind and the root cause can be identified more easily than if the fix is made later on. This has been emphasized in the following quotes:

> *"There are more time cost for doing analysis of the incidents and fixing them later."*

> *"It's a lot cheaper in terms of resources to fix the issue immediately rather than months down the track when they may be discovered"*

Participants all showed strong perceptions that TDD reduces the maintenance effort because developers are more confident to make changes to the existing application because they have a comprehensive set of tests that will let them know immediately if they "break" other functionality. This is implied in the following quote:

> *"With the legacy product, when incidents do come, we don't feel confident with re-factoring, and there is a lot time spend to test over and over again to make sure nothing breaks"*

Interviewees indicated that with constant running of automated regression tests inherent in TDD practice, developers can obtain instant feedback about whether the changes just made have been implemented as intended and if they break any old functionality. Moreover, participants indicated that this enabled developers to keep more in touch with their progress and have a more realistic view of their development status. As a consequence they tended to adjust their pace frequently when they knew they were behind the timeline.

Therefore, although TDD may result in longer coding time, most interviewees reported that TDD improves developer satisfaction and improves the overall development productivity. The quotation shown below emphasizes this idea:

> *"It is frustrating to write 15 tests but it's more frustrating to not be able to change something or not to know your changes are safe"*

Participants also claimed that tests are more meaningful in TDD and saw this as another factor contributing to a reduced maintenance effort. Developers reported being able to identify what functionality has been implemented more quickly, especially when working on code which was written by others or which hadn't been worked on for a long period of time.

In summary, the perception that productivity is improved in terms of end-to-end cycle using TDD was strong, although the observation was made that more time is spent on coding using TDD. The productivity gains are seen as coming from less time spent maintaining and making changes to the final application to fix errors. In addition an increase in productivity was seen as arising from the lower effort needed to understand code that hadn't been worked on for a while or was written by others. Participants also noted that using TDD improved developers' motivation and satisfaction, which may contribute to better developer productivity.

4.4 Conclusion

The research findings have been presented and discussed in this chapter. The case project and the backgrounds of the participants have been described to shows that these align well with the desired criteria. The results of this research have been analyzed and discussed to address the research questions.

By comparing the finding of TDD benefits in this study (as summarised in Table 4.2, Table 4.3 and Table 4.4) with the claims in literature (as summarised in Table 2.1, Table 2.2 and Table 2.3), it is evident that the themes identified from the interview data about the perceived benefits of using TDD align well with the three high level benefits claimed in literature. The contributing factors of those benefits indicated in literature have also been supported by the participants. Further insights into contributing factors to those benefits have been obtained from analyzing the participants' perceptions.

For example, participants claimed that TDD forces developers to analyze and develop small chunks of functionality and this contributes to the simple design. Participants also observed that this change in their mindset is the contributing factor for deeper and better understanding of the requirement and result in enhanced code and application quality. They also noticed that much of the information and semantics that were traditionally captured in comments became embodied in well written tests, and this contributes to better code quality and better developer productivity. Moreover, participants claimed that using TDD introduces increased developers' motivation and satisfaction. They noted that, with using TDD, developers are more confident about the functionality they produced as well as making any changes or code refactoring, and they are able to obtain a realistic status of the progress. These improvements result in developers being more willing to change and refactor to achieve better code quality and increase the overall development productivity.

Based on the results obtained from this research, a new framework of TDD benefits is developed, depicted in Figure 4.1. The new framework extends the prior framework developed in Chapter 2 by adding the contributing factors found for each claimed benefit.

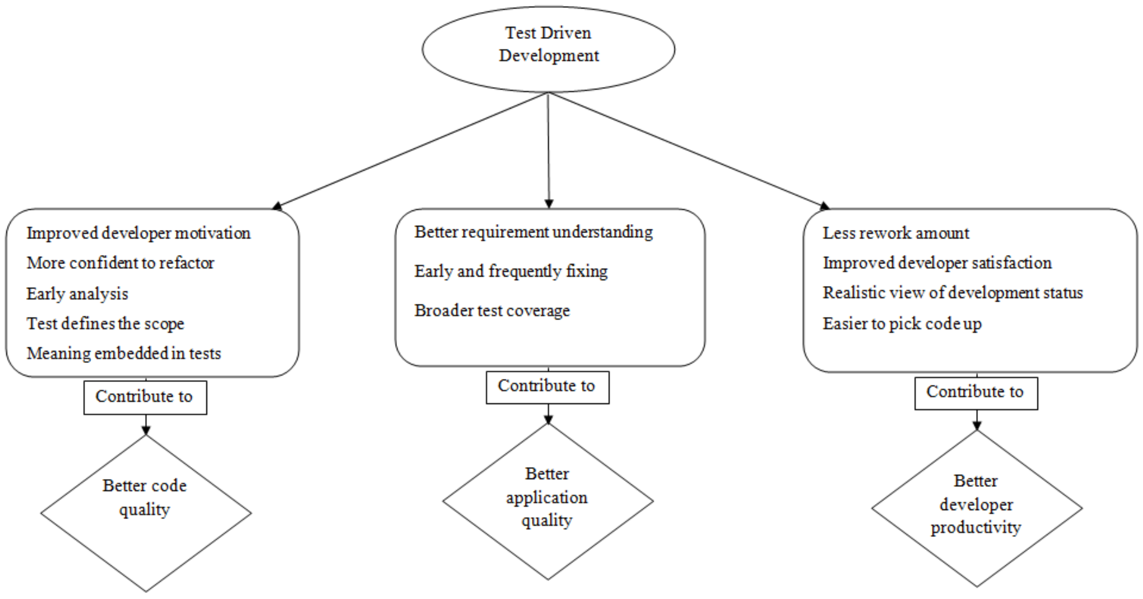

Test Driven Development
Improved developer motivation
More confident to refactor
Early analysis
Test defines the scope
Meaning embedded in tests
Contribute to
Better code quality
Better requirement understanding
Early and frequently fixing
Broader test coverage
Contribute to
Better application quality
Less rework amount
Improved developer satisfaction
Realistic view of development status
Easier to pick code up
Contribute to
Better developer productivity

Figure 4.1-High-level TDD Benefits and Contributing Factors

The findings gathered from the interview data also indicate some *shortcomings* of using TDD. Participants perceived that changing their way of thinking is difficult and time consuming. This same issue is reported in literature as well. Lack of tool support was also identified by participants as an issue with the use of TDD.

It is worth noting that, although the participants expressed their views on the benefits and causes quite strongly, this was generally based on their experiences gained over a long period of time, rather than quantitative evidence they had gathered.

The findings and discussion presented in this chapter are the basis for answering the research questions and developing the conclusion of this research. The next chapter provides the conclusion of this dissertation. It summarizes the main findings which have been drawn from this research related to the research objectives. The limitations of the study and possible future research are also discussed.

Chapter 5 – Conclusion

5.1 Introduction

In the previous chapter, the findings from the interviews were presented and discussed. This chapter presents a conclusion of the main findings of this research to answer the research questions (section 5.2). Furthermore, it discusses the limitations of this study in section 5.3, and this leads into the discussion of further research opportunities in section 5.4.

5.2 Conclusions of the Dissertation

In this section, conclusions regarding the main findings of this research are presented.

This research provides supporting empirical evidence that the same benefits claimed in literature are generally perceived as benefits by software development practitioners who have experience in both TDD and Test-Last development. Generally, the themes identified from the interviews data indicate that the participants' perceived benefits of using TDD align closely with the three high level benefits reported in literature, namely improved productivity, improved code quality and improved application quality. This provides answers to the research questions 1a and 1c.

Research question 2 is addressed by understanding the contributing factors to these benefits, drawn from the participants' perceptions. The findings of this study indicate that, by following the TDD process, there are changes to the developers' behaviours and attitudes in the software development process. Moreover, those changes are closely linked to the underlying factors that result in enhanced code and application quality, as well as improved developer productivity.

The findings in the research indicate that using TDD helps developers to think more about the requirements and boundary cases. A consequence of this is that developers obtain a deeper and clearer requirement understanding and produce an application with better code quality and fewer defects.

Participants also perceived that using TDD forces developers to consider small chunks of functionality during the development process. A consequence of this is an improvement in the code quality with simpler designs and lower coupling.

In addition, use of TDD encourages developers to provide semantically meaningful tests. This was perceived as a contributing factor to achieving “cleaner” code and easier code maintenance.

Participants perceived that using TDD encourages them to undertake more frequent and tighter testing, as well as providing them with immediate feedback. One consequence of this was described as the developers keeping in better touch with progress. Another outcome of this is that many problems were identified and fixed early during coding, resulting in fewer defects in the completed application. Quick feedback was also perceived as increasing developers’ confidence and willingness to refactor or extend code. This was seen as resulting in improved code quality.

Participants also noted that using TDD encourages developers to provide more extensive testing. This was identified as a factor that contributes to the enhanced application quality because the production code is well tested and likely to have fewer errors and be more robust. Participants also reported that this increases their confidence and satisfaction when delivering the application to the client, because they are more confident about the reliability of the application.

The interview data also provides some findings that address question 1b regarding the shortcomings of using TDD. Participants reported that the effort to change their way of thinking is difficult and time consuming. They also describe a lack of development tool support as another problem increasing the coding time with the use of TDD. They noted the “lack of integrated IDE support” as needing improvement.

Research question 3 relates to the participants’ satisfaction with TDD and how it was implemented, as well as their level of training. In general participants had a high level of satisfaction once the benefits of TDD were experienced. At the introduction of TDD practices the benefits were not clear to participants and they reported needing more effort to code and “unlearn” the Test-Last approach. To some extent the perseverance to continue with TDD seems to have come from (1) the persuasiveness and authority of the new project manager describing his positive experiences with TDD, (2) the difficulty in extending the legacy code with only a few tests available and the recognition that the process needed to change, and (3) the motivation gained with some early “successes” experienced as new code passed newly written tests.

Participants indicated that no formal training in the TDD process or TDD skills, such as writing quality tests, was provided. They described their learning as “on the job training” and noted that at the beginning they found it “hard to make good tests”. There was no clear opinion whether formal training in TDD would have helped to speed up the learning process or not. However, training in the way of thinking and analysis has been suggested by a few participants for improving the TDD practice.

In summary, this study provides some confirmatory empirical evidence supporting the efficacy of test driven development practice, as well as investigating the significant contributing factors. Figure 4.1 provides a useful analysis of the factors that contribute to these benefits to guide future research and practice. The findings of this research add to the existing empirical body of knowledge on the use of TDD in an industrial context, as well as strengthening the case for using TDD in this type of software development environment.

The results of this study should be of value to both the software development research community and practitioners.

The limitations of this research and suggestions for future study will be discussed in the following sections.

5.3 Limitations

Reflecting on the design and execution of this study uncovers some limitations this research. These are now discussed in this section.

One limitation of this study concerns the fact that each interview provides a viewpoint limited by the given representative’s perspective. It is possible that other roles within the company could have different points of view regarding test driven development that might enhance the richness and validity of the data. As stated in section 4.2, this research has not taken the views of certain roles related to software development, such as the project manager, who tends to obtain the domain knowledge from high-level management.

Another limitation relates to the fact that data were obtained from semi-structured interviews only, with no triangulation from other data sources. Experts (e.g. Tellis, 1997; Yin, 2003) recommend that “data triangulation” in case study research is important to improve the validity of the research. This usually consists of a comparison between data obtained from different sources such as interviews, document analysis and observation.

In addition, data sources for this research are participants’ perceptions collected through interviews, and it is possible for them to be interpreted differently to their original meanings by the researcher, thus threatening validity of the data. According to Yin (2003), relevant participants should review the data before they are recorded in the final report. However, due to time restrictions this activity was not conducted.

5.4 Future Research

Regarding the limitation discussed above, this study lacks input from the project manager group, an important role in the software development cycle. It is very likely that project managers’ perceptions would provide further insights into the practice of TDD. Thus, further investigation incorporating wider participation of roles can be considered as one possible future research area.

Similar empirical studies of practice incorporating wider range of software companies could be considered in a future study. This research gathered insights on TDD conducted in a product-driven software company. TDD implemented in a different type of software company, such as a software-as-a-service company, may have different outcomes. This may provide further useful insights and extend the findings in this area.

Moreover, reflecting on the discussion about TDD’s effects on developer productivity, participants expressed that “lack of IDE support” was a constraint on developers when programming using TDD. Therefore, further studies to investigate or develop new tools or techniques to improve this issue could be useful to practitioners.

References

Adler, P. A. and P. Adler (1998). Observational Techniques, in N. K. Denzin & Y. S. Lincoln (Eds.), Collecting and Interpreting Qualitative Materials. Thousand Oaks: Sage Publications.

Alliance, (2000). The Manifesto for Agile Software Development. Agile Alliance. Retrieved June 08, 2008, from http://www.agilealliance.org/

Auer, K. and R. Miller (2001). XP Applied, Reading,Massachusetts: Addison Wesley.

Beck, K. (2000). Extreme Programming Explained: Embrace Change, Reading, Massachusetts: Addison-Wesley.

Beck, K. (2001). Aim, Fire. IEEE Software 18(5), pp. 87-89.

Beck, K. (2003). Test-Driven Development by Example, Addison-Wesley.

Berg, B. L. (1998). Qualitative Research Methods for the Social Sciences, Boston : Allyn and Bacon.

Bhat, T. and N. Nagappan (2006). Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM.

Boyatzis, R. E. (1998). Transforming Qualitative Information: Thematic Analysis and Code, Thousands Oaks, London, New Delhi: SAGE.

Braun, V. and V. Clarke (2006). Using Thematic Analysis in Psychology. Qualitative Research in Psychology 3 77-101.

Canfora, G., A. Cimitile, F. Garcia, M. Piattini and C. A. Visaggio (2006). Evaluating Advantages of Test Driven Development: a Controlled Experiment with Professionals. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM.

Carver, J., L. Jaccheri, S. Morasca and F. Shull (2003). Issues in Using Students in Empirical Studies in Software Engineering Education. Proceedings of the Ninth International Software Metrics Symposium (METRICS'03), IEEE.

Collis, J. and R. Hussey (2003). Business Research: A Practical Guide for Undergraduate and Postgraduate Students New York: Palgrave MacMillan.

Crispin, L. (2006). Driving Software Quality: How Test-Driven Development Impacts Software Quality. IEEE Software 23(6), pp.70-71

Davis, A. M. (1995). 201 Principles of Software Development, New York : McGraw-Hill.

Denzin, N. K. and Y. S. Lincoln (2008). Collecting and Interpreting Qualitative Materials, Thousand Oaks, Calif : Sage Publications.

Erdogmus, H., M. Morisio and M. Torchiano (2005). On the Effectiveness of the Test-First Approach to Programming. IEEE Transactions on Software Engineering 31 (3), pp. 226-237.

Fox, N. (2006). Using Interviews in a Research Project. Retrieved July 07, 2008, from http://www.trentrdsu.org.uk/uploads/File/Using%20Interviews%202006.pdf.

George, B. and L. Williams (2003). An Initial Investigation of Test Driven Development in Industry. Proceedings of the 2003 ACM symposium on Applied computing. ACM.

Geras, A., M. Smith and J. Miller (2004). A Prototype Empirical Evaluation of Test Driven Development. Proceedings of the 10th International Symposium on Software Metrics, IEEE.

Godbole., N. S. (2004). Software Quality Assurance : Principles and Practice, Pangbourne, U.K. : Alpha Science International Ltd.

Gupta, A. and P. Jalote (2007). An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. First International Symposium on Empirical Software Engineering and Measurement, pp. 285 – 294.

Humphrey, W. S. (1989). Managing the Software Process, Reading, Massachusetts: Addison-Wesley.

Janzen, D. and H. Saiedian (2005). Test-driven Development Concepts, Taxonomy, and Future Direction. Computer 38 (9), pp. 43 - 50.

Janzen, D. and H. Saiedian (2008). Does Test-Driven Development Really Improve Software Design Quality? IEEE Software 25 (2), pp. 77 - 84.

Janzen, D. S. and H. Saiedian (2006). On the Influence of Test-Driven Development on Software Design. Proceedings of the 19th Conference on Software Engineering Education & Training (CSEET'06) IEEE.

Jeffries, R., A. Anderson, C. Hendrickson (2001). Extreme Programming Installed, Upper Saddle River, NJ: Addison Wesley.

Klein, H. K. and M. D. Myers (1999). A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. MIS Quarterly 23(1), pp. 67-93.

Lui, K. M. and K. C. C. Chan. (2007). Software Development Rhythms : Harmonizing Agile Practices for Synergy, Hoboken, N.J. : Wiley-Interscience.

Massol, V. and T. Husted (2003). JUnit in Action, Manning Publications.

Müller, M. M. and O. Hagner (2002). Experiment About Test-First Programming. IEE Proceedings - Software 149(5), pp. 131-136.

Myers, M. D. (1997). Qualitative Research in Information Systems. MIS Quarterly 21(2), pp. 241-242.

Nikula, U., J. Sajaniemi and H. Kalviainen (2000). A State-of-the-Practice Survey on Requirements Engineering in Small-and-Medium-Sized Enterprises. Retrieved March 21, 2009 from http://www2.lut.fi/~unikula/Publications/TBRCRR1.pdf.

Owen, W. F. (1984). Interpretive Themes in Relational Communication. Quarterly Journal of Speech 70, pp. 274-287.

Siniaalto, M. and P. Abrahamsson (2007). A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage. First International Symposium on Empirical Software Engineering and Measurement, pp. 285 - 294

Spinellis., D. (2006). Code Quality : The Open Source Perspective Upper Saddle River, NJ : Addison-Wesley.

Tellis, W. (1997). Introduction to Case Study. The Qualitative Report 3(2)

Valenzuela, D. and P. Shrivastava. (2007). Interview as a Method for Qualitative Research. Retrieved June 10, 2008, from http://www.public.asu.edu/~kroel/www500/Interview%20Fri.pdf.

Williams, L., E. M. Maximilien and M. Vouk (2003). Test-Driven Development as a Defect-Reduction Practice. Proceedings of the 14th International Symposium on Software Reliability Engineering.

Yin, R. K. (2003). Case Study Research: Design and Methods, Thousand Oaks: Sage Publications.

Appendix A – Interview Guide

Part1

Interviewee Experience

1. How many years experience do you have in Software Development?
2. How many software development projects you have done using TDD?
3. What training have you had in the use of TDD for software development?
4. What is your role in the TDD project?
5. Can you describe your understanding of what TDD means?

Project Information

6. Can you describe the details of the TDD project?

   (Probing questions if not covered in initial answer:

   6.1 How big the project is?

   6.2 What's the development environment for the project? Such as language and tools.

   6.3 What is the team size?

   6.4 What's the software development experience of team member?)

7. How TDD was incorporated in the project?

   (Probing questions if not covered in initial answer:

   7.1 What specific development processes were followed?

   7.2 What specific development techniques were adopted?)

8. Why do you think it was decided to use TDD in the project?

Part 2

Benefits of using TDD

9. What benefits do you think by using TDD in the project?

   (Probing questions if not covered in initial answer:

   9.1 Do you think TDD improves the code quality?

   9.2 Do you feel you can understand the requirements better with TDD?

   9.3 Did TDD contribute to better software (application) quality?

   9.4 Do you think you can develop faster with TDD?

   9.5 Do you feel TDD makes it easier for fixing issues?

   9.6 Do you feel TDD makes it easier when add functions to existing system?

   9.7 Do you think you are more motivated with using TDD?)

10. Can you please specify the evidence you have shows that using TDD provided the benefit?

    (Was something measured? Or was something different from not using TDD?)

11. Can you describe your understanding of why the benefit occurs?

    11.1 Do any TDD specific activities contribute to the benefit?

Disadvantages of using TDD

12. Did you find any disadvantages with using TDD in this project?

13. Can you please specify the evidence you have that using TDD was disadvantageous?

14. Can you describe your understanding of why the disadvantage occurs?

    14.1 Do any TDD specific activities contribute to the disadvantage?

Overall user satisfaction of using TDD

15. Would you recommend using TDD for other projects in the future?

16. Is there anything you would suggest to change using of TDD?

17. Would you like more training with TDD?

18. How widely do you think TDD is used in NZ? Worldwide?

19. What would you say the main differences between using TDD and non-TDD are for software development?

Appendix B – Participant information sheet

Participant Information Sheet

Date Information Sheet Produced:

16 February 2009

Project Title

Understanding the Efficacy of Test Driven Development

An Invitation

My name is Ling (Angela) Li. I am a student from AUT University, currently doing a research dissertation as partial fulfilment of a Master of Computer and Information Sciences degree. The research for this thesis is in the area of Test Driven Development. In particular it relates to understanding why Test Driven Development (TDD) provides benefits to software development.

Since this research involves understanding current practice, an essential element is partnering with practitioners in this area of software development. This invitation provides you with the opportunity to share your experience and perspectives and contribute to the body of knowledge in this area.

It is very important to note that your participation in this research is voluntary in nature, and you may withdraw your participation at any time without any adverse consequences.

The following questions and answers are intended to address the most common questions that the participant may ask about this particular research project. If you need further information, feel free to contact the researcher, Angela Li. My contact details can be found at the end of this document. It is recommended that you use e-mail to reach me.

What is the purpose of this research?

The purpose of this research is to investigate and better understand the main factors that make TDD an effective practice in software development. The primary focus of this research is to understand these factors from current practitioners' perceptions. This is important if TDD techniques and tools are to be improved, providing a practical basis for this. While the efficacy of TDD is well supported in literature, the underlying reasons for these incurred benefits have not been given much attention in research literature. Understanding current practice and perceptions will suggest possible process improvements and lead to the design of better support tools. This will ultimately contribute to faster and more accurate software development.

At the end of this research a report summarising the main results will be made available to you if requested. Furthermore, it is expected that some papers will be published in academic journals relating to this particular research project, with all information kept anonymous.

What will happen in this research?

You will be interviewed on a one-to-one basis with the researcher. This will be a loosely structured interview where you will be asked some open-ended questions related to your experience of TDD. The interviews will be held at your usual work places or any neutral place. The research will take some notes and also record the interview for later analysis. The analysis will involve coding and anonymising the data to identify trends and themes that provide insights to TDD.

What are the benefits?

As well as adding to the body of knowledge and influencing practice in this general area, the information will be made available to yourself and your colleagues and it is hoped that the knowledge gained will be useful for improving the practice in your organization.

How will my privacy be protected?

All of the materials related to the participants' information (consent form, tape, interview transcript, and interview note) will be stored at AUT in a locked drawer for at least 6 years. After that the material will be destroyed.

If you decide to withdraw from this research project for any reason it is guaranteed that all of the materials relating to you will be destroyed as soon as practicable after your request.

In addition, your employer will not have access to any interview records except when it is required under New Zealand law.

What are the costs of participating in this research?

The interview will take between .5 to 1 hours of your time. If a follow up interview is required, then a similar amount of time may be needed.

Will I receive feedback on the results of this research?

If you would like a report summarising the results of this research, please tick the appropriate box on the consent form.

What do I do if I have concerns about this research?

Any concerns regarding the nature of this project should be notified in the first instance to the Project Supervisor,

Jim Buchan
Senior Lecturer
School of Computer and Information Sciences
Auckland University of Technology
Private Bag 92006
Auckland 1142
New Zealand
Phone: + 64 9 921 9999 x 5455
Email jim.buchan@aut.ac.nz

Concerns regarding the conduct of the research should be notified to the Executive Secretary, AUTEC, Madeline Banda, *madeline.banda@aut.ac.nz* , 921 9999 ext 8044.

Whom do I contact for further information about this research?

Researcher Contact Details:

Ling (Angela) Li
Master of Computer and Information Science Lab,
School of Computer and Information Sciences
Auckland University of Technology
Private Bag 92006
Auckland 1142
New Zealand
Phone: + 64 9 921 9999 x 5410
Email zjs6868@aut.ac.nz

Approved by the Auckland University of Technology Ethics Committee on *27 April 2009*,

AUTEC Reference number *09/46*.

Appendix C – Participant consent form

Consent to Participation in Research

Project title: ***Understanding the Efficacy of Test Driven Development***

Project Supervisor: ***Jim Buchan***

Researcher: ***Ling (Angela) Li***

○ I have read and understood the information provided about this research project in the Information Sheet dated 16th Feb 2009.

○ I have had an opportunity to ask questions and to have them answered.

○ I understand that notes will be taken during the interviews and that they will also be audio-taped and transcribed.

○ I understand that I may withdraw myself or any information that I have provided for this project at any time prior to completion of data collection, without being disadvantaged in any way.

○ If I withdraw, I understand that all relevant information including tapes and transcripts, or parts thereof, will be destroyed.

○ I agree to take part in this research.

○ I wish to receive a copy of the report from the research (please tick one): Yes○ No○

Participant's signature:

....……………………………………………

Participant's name:

....……………………………………………

Participant's Contact Details (if appropriate):

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

Date:

Approved by the Auckland University of Technology Ethics Committee on 27 April 2009 AUTEC Reference number 09/46

Note: The Participant should retain a copy of this form.