Using Test-Driven Development to Improve Software Development Practices

Research Study
B.Sc Computer Science

RAQUELITA RÓS AGUILAR

Supervisor: Dr. Marta Kristín Lárusdóttir

School of Computer Science
REYKJAVÍK UNIVERSITY, SPRING 2016
T-622-UROP

Abstract

Background Software testing is the most effective method to evaluate and improve software quality (Orso & Rothermel 2000). Test-driven development (TDD) is considered to be one of those methods. This method is derived for the Agile method Extreme Programming (XP) and focuses on unit testing where developers incrementally write the tests before any code fragment and is considered to be a effective development approach. TDD has never been investigated in Iceland nor recieved special attention in the Icelandic software industry either, opposite to other countries.

Purpose This research is to investigate the pros and cons of TDD and to compare and analyze what Icelandic developers experienced when using TDD and when not using TDD. Empirical studies about TDD were found and their results were analyzed with the aim to discover what the advantages and disadvantages of TDD are.

Method This research study presents the results from seven semi-structured interviews with practitioners in Iceland working in the software industry who have adopted TDD in their projects. The interviews focused on the participants' point of view and experience of using TDD in software development. The data analysis of the interviews will be described along with a conclusion on how TDD can improve software development.

Results The results from this research show that TDD has the ability to help practitioners throughout the software development implementation phase in many ways. Other studies on TDD were analyzed with five factors in mind and their outcomes compared to this research which showed very similar results. The main advantages of using TDD can be outlined as: improved code quality, less defects, easier maintenance, safety net and reliable software.

Contribution The results will hopefully guide some software companies, managers and developers to understand the benefits and drawbacks of TDD and how to make a decision on whether or not to adopt TDD.

Keywords: Test-driven development, software testing, unit test, experience, software, quality, software coding.

Contents

Tables

Figures

Acknowledgement

I would like to thank Dr. Marta Kristín Lárusdóttir, my supervisor, for giving me feedback and great advices on how to conduct this research. I would like to thank the practitioners who were more than willing to participate in this research study and share their experience with me. Finally, I would like to thank my wonderful wife, Birna Guðmundsdóttir, for her endless support and love, and for reading over my research study when ever I needed a second opinion.

1 Introduction

Test Driven Development (TDD) is a software development approach that depends on short development cycles, by writing unit test first before any logical feature is implemented in a code. TDD is an approach originally from Extreme Programming (XP) which is considered useful software development methodology. Some may think that TDD has to do with testing and quality assurance, however TDD is a development / programming technique. It encourages developers to realise that their entire code needs testing, not just some parts of it. The table below shows the steps in the TDD cycle (table 1).

TDD Cycle		
No.	**Step**	**Result**
1	Add a test	Write a test for a specific feature according to requirements
2	Run tests and make the test fail	The new test can not pass without any code and therefor it must fail since the feature for the test has not been implemented
3	Write a code	Write **minimum** amount of code to pass the test
4	Run test	Run the tests and check if all pass. If any tests break at this point, then the code must be modified until the tests pass.
5	Refactor	When more features are added the code must be refactored and cleaned up frequently.
Repeat the cycle until feature has been implemented and tested		

Table 1: *TDD Cycle*[1]

When writing unit tests first, it can make sure that developers get the expected result, and eventually to write a code that don‘t break those tests. By adopting TDD developers are more likely to gain a diverse thinking on how to develop logic and algorithms in their code, and subsequently it can give them safety and confident in their work (Cannam, 2011). Therefor, one must assume that developers using TDD must be writing a better codes with less defects, and that must lead to a better product and user experience.

The purpose of this study was to find answers on whether or not there are benefits of using TDD in software development and if so, to provide some useful insights of adobting TDD. The research questions in this study are:

[1] https://en.wikipedia.org/wiki/Test-driven_development

1. What are the main advantages and disadvangates of using TDD?
2. Are there any savings when using TDD?
3. Is TDD helping developers in practice, if so in what aspect?
4. When should developers decide to use TDD?

The objectives of this research study is to investigate TDD based on active developers' experience and answer the questions above. Additionally, empirical studies about TDD were found and compared to the results of this research study.

2 Background

2.1 Motivation

A mandatory course at the University of Reykjavík (RU) introduces TDD roughly and it was in that course that the author first heard about TDD and became interested in the appraoch. The leading motivation for this research study is enthusiasm and interest in different work procedures, methods and methodologies that can lead to improve software development processes and productivity.The author wanted to look at what Icelandic software developers are doing to improve their work skills. Through the years, Icelandic developers have been seeking knowledge about TDD for themselves and some even created a small studygroup, around 2006[2]. The fact that TDD gets little attention in Iceland made the author want to investigate this subject more closely. The question is, can TDD achieve improvement in software development?

2.2 The Background of TDD

TDD is derived from the agile software development methodology called *Extreme Programming* (XP) and has only been around since in the mid nineties. Unit testing is the center of XP, where improved code quality and fast responsiveness for change in development is considered as some of the advantage of XP. This method demands, for example, comprehensive code review and unit testing the entire code[3]. XP focuses on pair programming where two developers work together using one screen, mouse and a keyboard. The developer actively working on the keyboard is sometimes called the *driver* and writes the code, while the other developer is sometimes called the *navigator* and is the one to come up with ideas and solutions. Every few minutes the developers swap roles.

XP methodology was created by Kent Beck which is the main advocate of TDD and one of the 17 software founders of *The Agile Manifesto*[4]. Beck claims that he did not create TDD and says that the original description of TDD is to be found in an ancient book about programming (Beck, 2012). This book said ,,you take the input tape, manually type in the output tape you expect, then program until the actual output tape matches the expected output.“. Nevertheless he is responsible for introducing TDD to software developers around the year 1996 along with XP. Before that in 1994, Kent had already created a powerful test framework called SUnit for the object-oriented programming language Smalltalk. Today, SUnit is

[2] Reference from participants
[3] https://en.wikipedia.org/wiki/Extreme_programming
[4] http://www.agilemanifesto.org

supported in many other programming languages like Java, Oracle and Python, just to name a few. For almost every common programming language there exists a unit test framework[5] which is designed to make unit testing easier.

As mentioned before, unit tests are the center of TDD but basic unit testing and TDD are not the same concept. Unit testing verifies a behaviour of a unit separately from other units in a code. With unit testing developers test single units, or methods, of their code with a test program that gives input to each unit, or method, and checks if it returns the expected output. It is a common misunderstanding that when using TDD all tests are witten before all the code. In TDD the unit tests are written before any code fragment is written. The tests outlines an agreement on what the code fragment is suppose to follow. That way, it supports that the code is written correctly and how the code is suppose to be designed. Therefore the main difference in TDD and basic unit testing is that TDD generates a condition which allows the tests to drive the development (Walther, 2009) while unit testing simply verifies a behaviour of a unit. Case study conducted by Williams et al. (2009) indicated that teams using TDD can reduce defects by 62% up to 91% compared to teams just using unit testing.

Despite what studies have shown, TDD is a very controversal appraoch and many have discussed how effective and practical it is, and even claim that TDD is simply a waste of time. Some developers have the impression that software testing is as expensive as the software development phase and therefore decide not to do any unit testing, while other developers are simply insecure about how to start unit testing.

Perhaps the most prominent debate on TDD was between some of the most public developers Kent Beck himself, David Heinemeier Hansen, the creator of Ruby on Rails web framework[6], and Martin Fowler, author and a public speaker on software development[7]. They had a online conversation about if TDD was dead. The root of this all was when David Heinemeier posted a blog in 2014 with the title *TDD is dead. Long live testing*[8]. Heinemeier was basically implying that developers should not put their faith in TDD because it could make them forget about testing the whole system, and by focusing on units it could result a poor system. Heinemeier also claimed that TDD was unhealthy and a bad idea to use on a code and stated this about his experience with TDD:

[5] https://en.wikipedia.org/w/index.php?title=List_of_unit_testing_frameworks&oldid=711724841
[6] Wikipedia - David Heinemeier Hansson
[7] Wikipedia - Martin Fowler
[8] TDD is dead. Long live testing.

> *"[…] at times I got sucked into that fundamentalist vortex, feeling bad about not following the true gospel. Then I'd try test-first for a few weeks, only to drop it again when it started hurting my designs."*

In their conversation, Beck responded to Heinemeier claims about TDD producing design damage by saying that it was like blaming a car for driving to a bad place. This conversation wen't on in five episodes on *Google Hangouts* where they tried to understand each others' views and experience[9]. They all agreed that TDD has value in some context, and the conversation ended with Martin Fowler saying:

> *"[…] if you're involved in software development you have to be thoughtful about it, you have to build up what practices work for you and your team, you can't take any technique blindly."*

This conversation got a lot of different reaction among software developers. One of them is known and respected software engineer and author, Robert C. Martin, or Uncle Bob as he is often called. He has written many articles and posts about TDD, created videos on how to use it, and is also one of the 17 software developers of *The Agile Manifesto*. Uncle Bob wrote an interesting post regarding Heinermeiers blog where he wondered about the credibility of Heinermeiers writing (2014). He made a lot of notable arguments in his post, for example regarding that Heinemeier called TDD a fundamentalism. Uncle Bob putted Heinermeier's choice of words in the context of 9/11 terrorist attacks and said:

> *"Fundamentalism is a term that used to mean: "back to basics"; but since 9/11 has taken on the connotation of violent extremism. I have yet to see any test-driven developers flying airplanes into buildings while repeatedly hollering: "Kent Beck is great!", so I must entirely reject the connotation."*

He ended the post by saying that he could be wrong but despite that he really trusts TDD for keeping the code clean.

It is interesting how software developers experience TDD differently yet many studies and articles have supported the effectiveness of this approach.

[9] Public conversation: Is TDD Dead?

2.3 Theoretical Background

Empirical studies on TDD show that adopting TDD in practice can have very beneficial effect on software development. There were four factors that caught the author most attention in other studies, that is; developers productivity, code quality, defects and maintenance. There are many possible benefits shown in other studies, but this research study will investigate these four factors further. In addition, one other factor will be investigated, and that is to see if TDD has any impact on developers confidence.

This section will show results from other research studies with these five possible beneficial factors in mind; developers productivity, code quality, defects, developers confidence and maintenance.

2.3.1 Developers Productivity

Some studies have shown that the developers‘ productivity when using TDD in practice can decrease because of the additional code writing for the tests. Bhat and Nagappan (Bhat & Nagappan, 2006) case study at Microsoft showed that the development time using TDD grew by 15% - 35%. Another study that was conducted two years later, generated by Nagappan et al. (2008) at Microsoft and IBM, showed the exact same numbers.

Williams and George (2004) showed that the TDD developers took about 16% more time to develop than other developers not using TDD. In that same experiment a survey was conducted where 78% of the developers believed that TDD actually improved the developers‘ productivity. Sanchez et al. (2007) longitude case study at IBM showed that the developers on the IBM team felt that TDD demanded more initial development time but it would be easier over time. Sanchez et al. said:

> *“The developers indicated there could be some productivity decreases, but the product lifecycle quality improvement would compensate for moderate perceived productivity losses.“*

Other studies that have investigated TDD differ from these results. Williams et al. (2003) conducted an empirical study four years before Sanches et al. (2007) which was also made at IBM. That study indicated that the productivity of the developers did not change by using TDD in practice. Janzen’s case study (2005) even showed that a team using TDD were more productive and implemented twice as many features compared to teams not using TDD, and with less effort per line-of-code. These studies show that there are some differences in the results regarding the developers’ productivity when using TDD.

2.3.2 Code Quality

Code quality is an importand factor in software development where in the long term high quality code is more useful and maintainable than low quality code. Sanchez et al. (2007) indicated that TDD reduces code complexity if used persistent over time. Williams and George (Williams & George, 2004) showed in their experiment that TDD deploys a code with superior external code quality, and 92% of 24 developers that participated in that experiment agreed with that result. Bhat and Nagappan (Bhat & Nagappan, 2006) indicated that code quality was 4.2 times greater with TDD and that TDD motivated developers to produce higher code quality.

2.3.3 Defects

Many studies have shown that using TDD reduces defects and gives less debugging time. Williams et al. (2003) empirical study at IBM was specifically investigating TDD as defect-reduction practice. The results indicated that by using TDD the developers spent less time debugging. Williams et al. stated:

> *"In this case study, we found that the code developed using a test-driven development practice showed, during functional verification and regression tests, approximately 40% fewer defects than a baseline prior product developed in a more traditional fashion."*

A study that Williams and George (2004) conducted, also supported that TDD reduces debugging, and 96% of 24 developers that participated in the experiment agreed with that. This indication is presented as well in Nagappan et al. (2008) results, where it was shown that TDD reduced defects between 40% and 90% compared to similar non-TDD projects.

2.3.4 Developers Confidence

Many developers who have used TDD claim that TDD verifies the correctness of their code, which gives them confidence as developers. Janzen's (2005) noted this in his study where developers reported higher confidence in making changes in their code if they were using TDD. Nagappan et al. (2008) stated that "[..] tests run should become the heartbeat of the system" and by constantly running unit tests it would provide developers a confidence to implement more features. Duka and Hribar (2010) agreed with this statement as shown in their paper about TDD. They noted that TDD ensured that all code is test covered which gives developers a greater level of confidence.

2.3.5 Maintenance

Reduction in software maintenance is also a favorable factor that studies have shown that TDD can produce. According to Duka and Hribar (2010) an easily maintainable software is one of the advantages of TDD. As mentioned earlier, a study from Nagappan et.al. (2008) showed that the development time using TDD grew by 15% to 35%, but they also showed that this increased development time was compensated for the reduced maintenance followed by a quality improvement. Mäkinen and Münch (2014) performed an literature review from prior empirical studies about the effectiveness of TDD and they all came to a similar conclusion. They noted that the maintenance phase quickly compensated the initial development time and that maintenance was in fact less time-consuming when using TDD.

2.3.6 Summary

Regarding the review above about possible benefits, the table below (table 2) shows summary of comparison where researchers analyzed how TDD affected those five factors. As shown in the table, four out of the five possible benefits increased when using TDD according to these studies.

Author	Year	Developers Productivity	Code Quality	Reduced Defects	Developers Confidence	Reduced Maintenance
Bhat and Nagappan	2006	Less	More			
Nagappan et al.	2008	Less		More	More	More
Williams and George	2004	Less	More	More		
Sanchez et al.	2007	Less	More			
Williams et al.	2003	Same		More		
Janzen's case study	2005	More			More	
Duka and Hribar	2010				More	More
Mäkinen and Münch	2014					More

Table 2: *Summary from prior studies regarding possible benefits of TDD*

3 Research Method

This chapter will explain the research method used for the interviews along with the participants' background.

3.1 Information Gathering Methodology

Interviews were used for information gathering in this research since they give access to the most qualitative data. The information that is collected through interviews can address significant factors when the right questions are asked.

To obtain greater understanding from the developers experience the prime approach was to use a semi-structured interviews for information gathering.

A semi-structured interview is when the interviewer strives to get desired information from a participant by asking prepared questions through an informal conversation. In these types of interviews the participants get to express their personal opinions and experiences on the subject and always have the option to change their responses. This type of information gathering has the ability to use open and direct questions on subject matter which can favor researchers greatly in their studies (Meehan, 2014). These qualities meet the requirements that this research study has for information gathering.

Each of the seven participants received an e-mail with an introduction to the reasearch study and an outline of the main goals of the interviews along with the structure of it (see Appendices A). In the interviews the participants were provided with prepared questions to look at, but they did only look at the list if they had any diffuculties understanding the questions asked. The questions related to the participants' software development background and experience of using TDD. The interviews were devided into five themes:

1) Background
2) TDD Background
3) TDD Project
4) Non-TDD Project
5) Closing Questions

Each interview took place where it suited the participants best and each interview took from 35 – 55 minutes. All interviews were recorded with a permission from the participants.

3.2 Selection of Participants

To be able to investigate the benefits of TDD and compare projects using TDD and without TDD the best solution was to interview a group of developers that had used TDD with some significant results, from their point of view. The participants were found through a Professor at Reykjavík University, contacts from interviewed participants and colleagues. It was not easy to find TDD developers here in Iceland and the result was that seven participants were interviewed from seven different software companies.

3.2.1 Participants´ Background

Each participant has a unique ID such as:

[P-1] [P-2] [P-3] [P-4] [P-5] [P-6]] [P-7], where P stands for participant.

The participants who were interviewed in this research study have been developing software in the range of 8 - 26 years and therefore development skills and experience varies. Five out of seven developers had over seventeen years of development experience with different companies over that time period (table 3).

Job Role	Education	Experience level	Age	ID
Developer	BSc and MSc	26 years	51	[P-1]
Developer and Scrum Master	BSc	20 years	40	[P-2]
Developer, Software Consultant and Instructor	BSc	17 years	42	[P-3]
Developer	BSc	8 years	39	[P-4]
Developer and Designer	BSc and MSc	17 years	33	[P-5]
Developer	BSc	20 years	39	[P-6]
Developer and Technical Producer	BA and MA	9 years	34	[P-7]

Table 3: *Participants' background*

The participants are currently working for a companies that are for example developing softwares such as backend solutions for airline companies, software as a service on the internet, web systems, internal systems for insurance companies, browser, computer games and also as a software consultant. The number of developers working in the participants companies are from 1 up to 50 people.

Numbers of developers	Quantity	Participant
1-20	5	[P-1][P-3] [P-4] [P-5][P-6]
21-30	1	[P-2]
31-50	1	[P-7]
Total	**7**	

Table 4: *Number of developers in each participant comanpy*

3.3 Data Analysis

All recordings from the interviews were listened to and the data was compiled to a Word document. The data was then reviewed and the participant's responses were categorized by each question. Notes and ideas were made after the first review and relevant issues were coded manually. The codes were then grouped together and themes identified.

4 Results

This chapter shows the results from the data collection from the interviews and the participants' background on TDD along with software projects they have developed with TDD and without TDD.

4.1 TDD Background

None of the developers had ever heard of TDD before the year 2003. When they first noticed TDD, six[10] of the participants were really fascinated and interested in learning more on how to adopt TDD. After using TDD for some time all of the developers were unanimous about their impression being really positive and some were even more impressed than before with this approach. Four developers[11] mentioned that they soon found out that TDD had much more to offer than they thought originally. TDD had changed their thinking in general regarding programming and software developing. All of the participants stated that TDD is all about practice, even though it is easy to learn TDD, it can take many years to find the correct path. One developer describes this as:

> *,, I look at TDD as an investment which took me many years to achieve where I am today. It took me some years to get my TDD-nirvana.*[12] *"*

4.1.1 TDD Experience Level

Three participants have been using TDD for over 10 years (table 5). None of them had ever heard about TDD in universities and had to seek knowledge for themselves after they started working as a developers. They found most of the TDD material online, like videos with Uncle Bob[13]. None of the participants thought it was hard to learn TDD but said that it took a lot of discipline and mindshift, and a whole lot of practice. The table below (table 5) shows the participants' experience level using TDD.

Using TDD	Quantity	Participants
2 – 5 years	2	[P-4][P-7]
6 – 10 years	2	[P-1] [P-2]
11 – 15 years	3	[P-3] [P-5] [P-6]
Total	**7**	

Table 5: *Number of years using TDD*

[10] [P-1] [P-3] [P-4] [P-5] [P-6][P-7]
[11] [P-3] [P-4] [P-6][P-7]
[12] [P-6]
[13] https://en.wikipedia.org/wiki/Robert_Cecil_Martin

4.1.2 Advantages

The participants were asked to briefly list up from their own experience what advantages they thought were derived from using TDD. The main factors that appeared more than once from the participants' lists can be found in the table below (table 6).

Advantages	Number mentioning	Participants
Improved code design and code quality	5	[P-1][P-3] [P-4] [P-6][P-7]
Easier and faster to make changes	4	[P-2][P-3][P-5][P-7]
Easier to add new feature	4	[P-2][P-5][P-6] [P-7]
Reduced maintenance	4	[P-2][P-3] [P-5][P-6]
Reduced defects	4	[P-3][P-5][P-7][P-7]
Safety net	3	[P-1][P-5][P-6]
Loosely coupled code	2	[P-1][P-4]
Code documentation	2	[P-2][P-3]

Table 6: *Benefits of TDD according to participants experience and numbers of participants mentioning those*

4.1.3 Disadvantages

There were not many disadvantages of using TDD according to the participants. However there was one factor that four participants[14] mentioned, and that was: "complicated and badly written unit tests which can result in extremely fragile tests". Evidently, it is easy to develop these kinds of fragile tests if the scope is not well defined for each unit in the system. According to them, bad unit tests are for example tightly coupled code with lot of dependencies. One participant[15] mentioned that unit tests can sometimes have the ability to push developers to develop a system mainly around certain units which can often lead to complexity instead of simplicity. He said:

> *"Testable units are not necessarily better in all cases. With TDD you need to let the software design serve the tests, which I don't think is the best thing to do for all projects."*

[14] [P-1] [P-2] [P-3] [P-5]
[15] [P-5]

Furthermore, he said that sometimes when changes are made in a code, it could be hard to maintain the tests as they tend not to behave correctly and break easily after. In contradiction with that, other participants stated that if the tests do not behave correctly after changes, it's because the tests are badly written and the responsibility lies with the developer not the approach. When asked more about badly written tests, a few of the participants said they were common among inexperienced TDD-developers.

4.1.4 Time Consuming

None of the participants felt like using TDD was time consuming. Although two participants mentioned that the only time they felt like it was a waste of time writing unit tests was when they were badly written, which had nothing to do with TDD. They noted that in the long run they see profits when using TDD. Following that, one said:

> *"I feel bad every time I don't use TDD. You get much more confident with the tests and can make changes without fear.*[16]*"*

All participants noted that because of writing the unit tests first the initial time cost was greater when using TDD, but definitely worth every minute. One participant[17] said that when using TDD the tests allow him to move on, which he considers to be a safety net. Another participant[18] mentioned that when he first started using TDD and felt like it was a waste of time, he would be tempted to write the tests last. But after using TDD for some time he became aware of what it can return, and realized that writing tests first was unquestionably time well spent.

4.1.5 Types of Projects When Using TDD

When asked if they used TDD in all projects, five participants said that they do not use TDD in all of their projects while two said that they always use TDD. When asked what type of projects they do not use TDD in, all of the five participants said they usually do not apply TDD to an old code, particularly if the code was written by other developers, and codes that have no documentation such as unit tests. All participants always use TDD when they are developing a new project or rewriting an old system from scratch.

[16] [P‘7]
[17] [P-6]
[18] [P-4]

4.1 TDD vs. Non-TDD

Each participant was asked to describe and answer questions about two projects that they already have developed (see Appendices C). The first project was developed with TDD while the second project was developed without TDD. For each participant the result from those two projects were compared to see the main difference from their experience.

The scales from the participants' team and project size were different. The aim of this research study is mainly to observe what the benefits and drawback of TDD are, and by looking into dissimilar projects, the effectiveness of TDD will be revealed further.

4.1.1 TDD Projects

The table below (table 7) demonstrrates overview for each participants TDD project containing the main factors.

	Project 1	**Project 2**	**Project 3**	**Project 4**	**Project 5**	**Project 6**	**Project 7**
Description	Client for browser	Backend	Server	Internal system	Web service	Internal system	Distributed game system
Participant role	Developer	Developer	Developer	Developer	Developer	Developer	Developer
Team size	2	5	20	6	1	6	4
Project size	18 months	6 months	12 months	10 months	2 months	24 months	3 months
Programming language	Javascript	Java	Java	Javascript	Java	Javascript	Python
Methodology	Kanban	Scrum + Kanban	Scrum	Scrum + Kanban	Kanban	Scrum	Scrum

Table 7: *TDD project - Overview*

As shown in the table above the size of the teams runs from 1 to 20 people. All the participants were developer. The projects size runs from 2 to 24 months, where some of the projects are ongoing projects but the participants had completed functional and deliverable features from it. When asked why it was decided to use TDD in these projects, each participant replied that it was a decision based on prior experience of using TDD. All of the seven projects were either a new system, new independent unit or a replacement of an older system, and therefore they felt it was ideal to use TDD.

4.2.1.1 Problems

The partipants were asked if any problems occurred during the development phase and if TDD helped them to solve these problem. Six participants responded positively. It was pointed out that by using TDD it was easier to catch bugs and defects earlier in the development phase and as a result, TDD supports continous deployment better than otherwize. Subsequently, TDD gave the developers more confidence in their work and they felt like TDD made sure that they could develop softwares with safety.

4.2.1.2 Development Methodologies

All of the teams were following an Agile software development methodologies, either Scrum or Kanban. In project 2 and 4, Scrum was applied during the development phase and Kanban during the test- and maintainance phase.

4.2.1.3 Using TDD - Advantages

The participants were then asked about the benefits of using TDD in these projects and only to respond with *yes* or *no* to the elements shown in the chart below. The participants' replies to these questions, related to both these specific projects in question and general in software development.

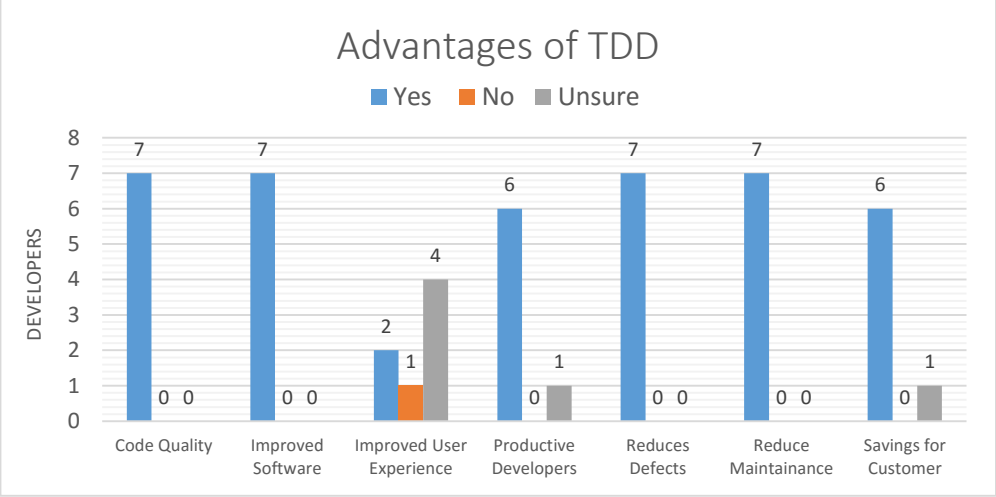

Figure 1: *Benefits of using TDD in software development*

Code Quality and Productivity

All participants were confident that improved code quality and software were one of the benefits of using TDD. When asked about the developers productivity during the project life cycle, only one participant said that he thought it is was hard to tell and felt like he could not answer either yes or no to that question. Six participants[19] said that the productivity evens out through time, wherein one of them said:

> *„With TDD you get a predictiable pase. Developers start slower but in the end they spend less time in regression and fixing bugs, therefore it evens out through time. Following that, software maintenance gets easier and faster with TDD.*[20]*“*

Savings

Reduced defects and maintenance were also benefits that all participants had experienced using TDD. According to them TDD generates a loosely coupled code thus maintenance gets cheaper, easier and faster. When asked if there were any savings for a customer / product owner when developing with TDD six participants believed that the savings lied definitely in the maintainance phase. One participant could not answer this question with certainty since the project in question was developed for the company itself and the customers only paid monthly fee for their registered account in the software.

UX

Three of the participants believed that TDD does not affect user experience (UX) but after thinking about it two of the participants changed their answer to yes because they believed TDD reduces defects. They said if using TDD catches bugs as early as in the develpment phase and before the system is delivered to end users, then it must improve the user experience. Four participants were unsure whether TDD had any effect on user expersience.

The summary of their understanding of why these advantages took place was because by adopting TDD they believe one has a reliable and a safer software that supports changes and desired requriements, as shown in Figure 1.

[19] [P-1] [P-2] [P-3] [P-4] [P-5] [P-6]
[20] [P-7]

4.2.1.4 Using TDD - Disdvantages

Finally, the participants were asked if there were any disadvantages of using TDD in these particular projects. Four of the participants said they did not believe that using TDD in their projects had any disadvantages. One said that not all team members syncronized regarding writing the tests first and that affected the development phase, and one said that it took some time to write all the tests and get started on the project itself.

4.1.2 Non-TDD Projects

The table below (table 8) demonstrates an overview for each participants' non-TDD project containing the main factors.

	Project 1	**Project 2**	**Project 3**	**Project 4**	**Project 5**	**Project 6**	**Project 7**
Description	Data Analyzer	Displaying data in a client	Server	Internal system	Web service	Microsoft Plugin	Calculator for distributed system
Paricipants role	Developer	Developer	Developer	Developer	Developer	Developer	Developer
Team size	12	3	20	8-10	1	1	3
Project size	5 months	3 months	24 months	16 months	3 months	12 months	36 months
Programming language	Java	Java, PHP, Flash	Java	Python	Java	C++	Python
Methodology	Waterfall	None	Scrum	Scrum	Kanban	Waterfall	Kanban

Table 8: *Non-TDD project – Overview*

As shown in the above table the size of the teams runs from 1 to 20 people and the role of the participants were developing the projects. The projects size runs from 3 to 36 months where some of the projects are ongoing projects but the participants had completed a functional and deliverable features from it, similar to the TDD projects.

4.2.2.1 Problems

The participants were asked if any problems occurred during the development phase and / or delivery of the product, and only one participant[21] did not experience any problems. The other six participants all experienced some kinds of problems which are shown in the table below (table 9).

Problem	Participant
Difficulty finding the bugs	[P-3][P-6][P-7]
Lot of regression defects	[P-1][P-7]
Layers did not work together	[P2]
Dry errors and bugs	[P-6]
Considerably long time to verify if the code were correct	[P-4]
Very hard to maintain some of the units	[P-3]

Table 9: *Non-TDD project - Problems during the development*

4.2.2.2 Development Methodologies

Four teams[22] followed Agile software development methodologies, that is Scrum and / or Kanban, two teams[23] followed the Waterfall methodology, which they did not find successful, and one team[24] did not follow any methodology. When asked if any formal testing methods were used throughout the implementation, three participants replied negatively, three said that a software tester did some intergration tests and one said that their company used Selenium plugin tests which was applied by a software tester after most of the features had been implemented.

4.2.2.3 Without TDD - Benefits

The participants were asked if they thought there were any benefits of not using TDD in their projects. All the participants replied negatively and believed that it would have been beneficial for the project to have the developers adobting TDD. They were then asked to explain why they thought that, and some responded that TDD makes units testable and maintainable throughout the development and further along. Others responded that the code design and code quality would have been better with TDD and it would have been a great benefit to have automated tests since TDD supports that very well. One[25] mentioned that acceptance

[21] [P-5]
[22] [P-3] [P-4] [P-5][P-7]
[23] [P-1] [P-6]
[24] [P-2]
[25] [P-2]

test-driven development (ATDD) would have been the best choice for that project at that time. For all teams, no formal tests were made during the development phase.

Since the interviews were semi-structured it gave the participants an oppurtunity to speak freely. As a result it was easy to find whether the participants thought there were any benefits of not using TDD in these projects, and general in software development. Presented in the chart below is the outcome from that data analysis (figure 2).

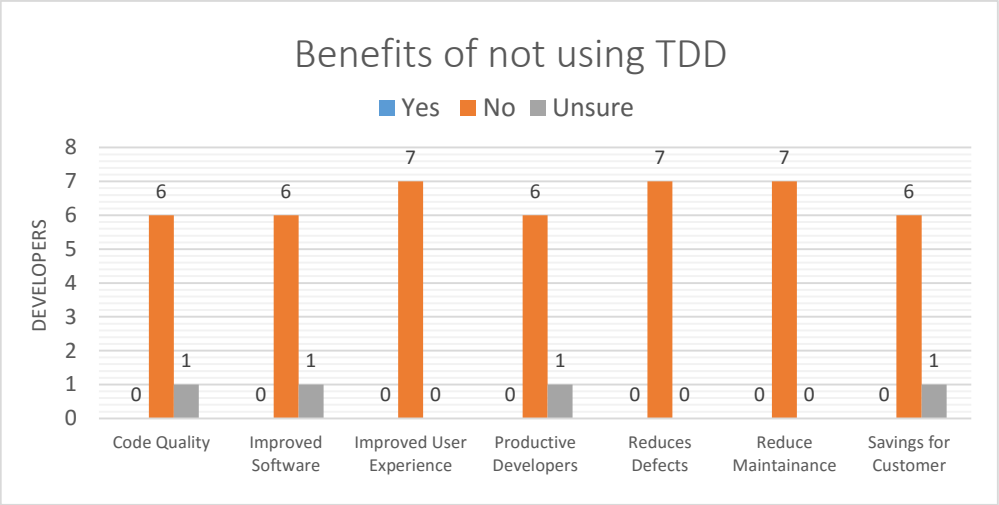

Figure 2: *Benefits of not using TDD in software development*

4.2.2.4 Reason for not using TDD

Finally the participants were asked why TDD was not used in these projects. Six participants[26] said the reason was mainly because they did not know TDD at that time, while one participant[27] said that it was the developers' joint decision to not adopt TDD for that project.

[26] [P-1] [P-2] [P-4] [P-5] [P-6][P-7]
[27] [P-3]

4.2.3 Closing Questions

Three general questions about TDD were asked at the end of each interview (see Appendices D). When asked, all participants responded that from their own experience they would definitely recommend TDD for others to use in the future.

4.2.3.1 Teach TDD in Universities

The participants were then asked if they thought that universities should put more emphasis on teaching TDD in computer science courses. All participants responded that they felt that TDD deserved more attention in universities. Details of their responses are shown in the table below (table 10).

Comments about teaching TDD in universities	Quantity	Participant
There should definitely be more emphasis on teaching TDD	7	[P-1] [P-2] [P-3] [P-4] [P-5] [P-6][P-7]
I think that TDD should be a recognized practice for developers when they graduate from computer science	3	[P-1] [P-2] [P-6]
The methodology and the discipline thinking that is behind TDD should be tought in universities	2	[P-2][P-7]
Teaching XP and pair programming would be very smart to do and would give students advanced knowledge	1	[P-1]
Students should know how to use TDD with the tools that supports the tests in TDD	1	[P-2]
I think I would have been more prepared for my professional career and I also believe that I would have developed a better product if I had known TDD	1	[P-4]
I would have wanted to take at least one mandatory course in TDD when I was studying computer science	1	[P-4]
TDD derives better developers and teaching TDD in universities supports that	1	[P-1]
When I think about it, TDD takes a lot of practice so I don‘t know how much students can learn or earn skills in school. But they should indeed recognize TDD from their studies	1	[P-6]
I think that it would be great if teachers encourage (not insist) students to be test-driven and even give them extra points for that	1	[P-6]
TDD is all about practice. I think that if students would start early to learn how to use it, they would get more experience throughout the university	1	[P-7]

Table 10: *Participants comments about teaching TDD in universities*

As shown in table 10 all of the participants agreed that TDD should be more recognized in universities and they also believed that focusing more on TDD in some courses would have a positive effect on students.

4.2.3.2 Difference between TDD projects and non-TDD projects

Ultimately, the participants were asked to distinguish the differences of using TDD and not using TDD in software development. As a result the following table (table 11) contains a list of what the participants did experience when using TDD in practice.

Ruling differences of using TDD and not using TDD in software development	Quantity	Participant
With TDD you can detect and prevent bugs earlier	3	[P-1] [P-2] [P-3]
Better code design with TDD	3	[P-3] [P-4] [P-6]
TDD helps developers achieve maintainable code	1	[P-3]
TDD provides a great safety in the development	1	[P-4]
With TDD it is much easier to change and add a feature to a system	1	[P-5]
TDD generates a loosely coopled code	1	[P-6]
Less maintenance after delivery if you are test-driven	1	[P-6]

Table 11: *Ruling difference of using TDD and not using TDD*

4.2.3.3 Speak Freely

To finish the interviews each participant was asked if there was something they would like to add that they thought was relevant for this research study. The participants' answers regarded TDD, BDD[28], and unit testing. Two participants[29] said that sometimes they use BDD instead of TDD, and that it has been argued which one is better in practice. One of them said that BDD had been working fine for him but not great, and mentioned that it is much harder to refactor and change the code with BDD.

Unit testing received the most attention in the participants' comments. Five participants[30] agreed that developers should always do unit testing, whether they write them first or last, one participant said:

[28] Behaviour Driven Development
[29] [P-1][P-6]
[30] [P-1] [P-3] [P-4] [P-5] [P-6]

> *„There are many who use unit tests but do not believe in TDD. While other say that TDD is the silver bullet that fixes everthing, I don‘t agree with that.* [31] *“*

Some of the participants talked more about TDD where one participant[32] mentioned that TDD was a very good approach but that it could not replace years and years of experience in programming. Another participant said that he looks at TDD as a craft and a skill, and said:

> *„Once you get to the skill level of using TDD, I don‘t see any reason why you should ever skip using TDD.* [33] *“*

Finally, one participant said that TDD is a work-approach which he follows, and indicated that it would be a deal-breaker if he would apply for a job that does not support TDD in practice.

[31] [P-5]
[32] [P-3]
[33] [P-6]

5 Discussion

This chapter will show answers to the research questions along with analysis from the interview data. It will also show if any similarity are with the results from this research study and prior studies referred to in chapter 2.

5.1 Research Questions

RQ1: What are the main advantages and disadvantages of using TDD?

During the interviews it became apparent quickly that the participating developers felt that the advantages of using TDD were significantly more then the disadvantages, and were actually having difficulties finding what the disadvantages were. Based on their experience, TDD can be very effective when used correctly and conscientiously.

Advantages

This research study supports prior researches of what the main beneficial factors of TDD are. The results in this research are showing that TDD supports developers to write a code with better quality which produces a code that is more responsive to changes and to new features. The participants stated that by writing tests first and creating a test suite to validate the correctness of their code, they experience less defects, easier refactoring and less maintenance. It can be concluded from the results that TDD provides a great safety for developers which gives them motivation in their work.

Disadvantages

60% of the participants said that badly written unit tests were the main disadvantage of TDD, but stated that it was not the approach that produces bad and fragile unit tests, rather the developers themselves. During this study one of the expected disadvantage was that developers felt that writing tests first was time consuming and slowed them down. The results did not support that and showed that the developers realized that the benefits of using TDD balances the time they spent on writing the tests, and often leaves them feeling more productive throughout the project life cycle. It was mentioned that when developers are inexperienced with TDD they go through a certain learning phase that can be hard and tedious at times which some may take as a disadvantage to begin with. Despite all of that, the participants believed that TDD is worth all the hard work.

RQ2: Are there any savings when using TDD?

Because of the additional time in writing unit tests some may be convinced that it is too expensive to adopt TDD. According to participating developers they believed that there are in fact savings when using TDD. With reduced defects and easier maintenance they consider that the savings lies mainly in the maintenance phase, and believed that TDD has qualities that can lead to improved software. It can be concluded that the improvements that TDD derives have the ability to bring a notable saving in the project life cycle cost.

RQ3: Is TDD helping developers in practice, if so in what aspect?

According to the participants, by adopting TDD they had to go through change of attitudes to get into the so called TDD mindset. Besides the advantages mentioned above, TDD has given the developers deeper logical understanding of their code, and as a result it has supported them to improve their development skills. In general the participants felt more confident in their work and more prepared to handle any changes on a code when using TDD.

RQ4: When should developers decide to use TDD?

From the participants' feedback it was indicated that when the they decided to adopt TDD it was mainly because they were looking for a way to improve themselves and their work. They all agreed on the fact that TDD had improved their work and even motivated them to keep improving. All participants recommended TDD when starting a new project or rewriting an old system.

Based on the interview results developers and companies should decide to use TDD if they want to:

- improve code quality and code design
- produce less defects and bugs
- have a reliable software
- make changes easier and faster
- make maintenance easier and faster
- support continuous delivery
- experience more confidence in their their work

These are the motivations as to why the participants still use TDD to this day.

5.2 Developers Productivity

Although, at least four prior research studies in chapter 2 are showing that developers' productivity decreases when using TDD, the results in this research are in line with Williams et al. (2003) and Janzen's (2005) findings on developers' productivity. These two prior studies demonstrated that TDD did not slow developers down and some even felt more productive. The interviews in this research are showing that even though the initial development time can be greater, TDD has the ability to speed up the process after that. The results indicate that developers encounter improved productivity with increased TDD experience.

5.3 Code Quality

Improved code quality leads to more desirable advantages, such as easier code maintenance and less defects. The empirical studies that were analyzed in this study all show that improved code quality is much greater with TDD. Bhat and Nagappan (2006) showed that developers using TDD are more motivated to produce higher code quality. This study is showing similar results, that developers experience improved code quality and code design when using TDD. The participating developers believe that this improvement is one of the main benifical factors of TDD.

5.4 Defects

The results in this study is showing that when using TDD, developers detect and prevent defects earlier in the development. These findings are in compliance with other TDD studies. Williams et al. (2003) and Nagappan et al. (2008) showed in their findings that TDD has the ability to reduce defects by 40 - 90%. Other empirical studies have shown similar results. The results in this study are showing that by writing tests first and assure the correctness of the code, developers gain better understanding of the software requirements which leads to less defects.

5.5 Developers Confidence

As discussed in previous sections, confidence is a factor that participating developers had experienced from TDD. Nagappan et al. (2008) said that by constantly running unit tests, developers gain great confidence in their work. These findings are perfectly in line with the results in this research study. Participating developers stated that TDD produces tests that support every logic and functionality of a software, that support gives developers confidence to make changes or add new features to their software.

5.6 Maintenance

Software maintenance can, at times, be hard and demanding for developers. The results from this research study shows that participating developers experienced great reduction in the maintenance phase when they used TDD. With the test coverage that TDD generates, developers spent less time and effort in debugging, making the maintenance much easier. These results support other studies, such as Nagappan et.al. (2008), and Mäkinen and Münch (2014) studies. They claimed that TDD reduces software maintenance, as developers felt it was easier and faster to maintain a software that was developed with TDD. It was also shown that even though developers spent more time on writing unit tests with TDD, that time was regained with reduced and easier maintenance, as the results from this research study show. It has been indicated that TDD has the potential to produce a high quality and modular code, a code that tends to be extremely maintainable. The above results are the consequence of improved code quality that TDD is believed to produce.

5.7 Summary

The table below (table 12) shows the summary of comparisons where researchers analyzed how TDD affected those five factors above along with the results from this research study.

According to these studies four out of five possible benefits increased when using TDD, as shown in the table below. It is interesting to see how all the studies found that code quality, reduced defects, developers confidence and reduced maintenance increased while most of the studies showed that developers productivity decreased when using TDD. Two of the studies showed the developers productivity to be more, and two found it to be same. Perhaps it would be worth the effort to start trying to change the mindset of computer science students right at the beginning of their studies so that they don't feel like writing unit tests is a waste of their time.

Author	Year	Developers Productivity	Code Quality	Reduced Defects	Developers Confidence	Reduced Maintenance
Bhat and Nagappan	2006	Less	More			
Nagappan et al.	2008	Less		More	More	More
Williams and George	2004	Less	More	More		
Sanchez et al.	2007	Less	More			
Williams et al.	2003	Same		More		
Janzen's case study	2005	More			More	
Duka and Hribar	2010				More	More
Mäkinen and Münch	2014					More
Aguilar	2016	Same / More	More	More	More	More

Table 12: *Comparisons from other studies and this research study*

5.8 TDD vs. Non-TDD

The two projects and teams size that each participant was asked to describe were dissimilar, but the aim of that focus was not to compare the projects themselves but rather to find out if developers experienced differences in outcomes of various projects. It was interesting to see that the main reason the participating developers did not use TDD in the Non-TDD projects was mainly because they did not know TDD at that time. The results indicate that when developers experience and realize how effective this appraoch is, they employ it whenever they can.

In the TDD projects participating developers believed that TDD helped them solve many problems that occurred during the development, problems they experienced in the Non-TDD projects, such as regression defects and difficulty locating bugs. According to the results, these problems, and others, could have been prevented, or at least minimized, if TDD had been applied. The results also show that all teams in the TDD projects followed either Scrum or Kanban but only four teams in the Non-TDD projects. It appears that when developers apply TDD in their practice they are more likely to become Agile. These results are perhaps not surprising since TDD is derived from the agile software development metholodgy XP. In summary, the results show that participating developers were more satisfied with their work when they used TDD.

5.9 Limitations

All data in this research study is the developers‘ experience using TDD, therefore it was not possible to gather quantitative evidence about how effective TDD can be. It would have been interesting to collect quantitative data, however it is very hard to measure the effectiveness of TDD.

6 Conclusion

The results from this research study show that adopting TDD in practice can be very beneficial when used correctly. Although this research study and other studies show favorable results when using TDD, developers must have the correct mindset when using TDD, which requires great discipline, a quality not everyone has. It must be noted though, that TDD is not just about writing tests first and creating as much test coverage as possible. TDD is a development approach that drives the design of a code and supports developers in their work with confidence and more reliable software.

As shown in this study, participating developers believe that TDD should receive more attention in Iceland. It was not easy to find developers that had been using TDD with significant results and as this research study shows, participating developers recommended that universities put more emphasis on teaching TDD. They stated that TDD derives better developers, and would give students greater experience which could prepare them better for their professional career. Considering how difficult it was to find developers that are using TDD makes one wonder if it is because they think TDD is really just a waste of time or because developers in Iceland are mainly unfamiliar with this approach. The results in this research study show that developers who use TDD are in general really satisfied with the approach, which is in line with other studies.

The findings in this research study provide supporting evidence to prior research studies on how effective and useful it can be to adopt TDD in software development.

7 Future Work

After working on this research study many questions and ideas arose on how to explore the effectiveness of TDD. The author encourages other researchers to investigate TDD, for example by conducting a case study that measures the effect of TDD by comparing teams, developers and / or projects that are using TDD and not using TDD and count the bugs and defects over a period of time. The author also thinks that it could be interesting to investigate TDD teaching approach that would be applied in higher education, and whether or not it is easy to understand and adopt TDD.

References

Beck, K. (2012). (4) Why does Kent Beck refer to the “rediscovery” of test-driven development? - Quora. Retrieved February 19, 2016, from https://www.quora.com/Why-does-Kent-Beck-refer-to-the-rediscovery-of-test-driven-development

Bhat, T., & Nagappan, N. (2006). Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies. Retrieved from http://www.msr-waypoint.com/en-us/groups/ese/fp17288-bhat.pdf

Cannam, C. (2011). Unit testing: Why bother? | Sound Software .ac.uk. Retrieved April 8, 2016, from http://soundsoftware.ac.uk/unit-testing-why-bother/

Duka, D., & Hribar, L. (2010). Test Driven Development Method in Software Development Process. Retrieved April 13, 2016, from https://bib.irb.hr/datoteka/483416.TDD.pdf

Janzen, D. S. (2005). Software Architecture Improvement through Test-Driven Development. Retrieved April 11, 2016, from http://src.acm.org/subpages/gf_entries_06/DavidJanzen_src_gf06.pdf

Mäkinen, S., & Münch, J. (2014). Effects of Test-Driven Development: A Comparative Analysis of Empirical Studies. Retrieved April 13, 2016, from https://tuhat.halvi.helsinki.fi/portal/files/29553974/2014_01_swqd_author_version.pdf7

Martin, R. C. (2014). Monogamous TDD | 8th Light. Retrieved May 23, 2016, from https://blog.8thlight.com/uncle-bob/2014/04/25/MonogamousTDD.html

Meehan, K. (2014). Interviews, Semi-Structured - Qualitative Methods in Geography - wiki.uoregon.edu. Retrieved February 20, 2016, from https://wiki.uoregon.edu/display/qualmethodsgeog/Interviews,+Semi-Structured

Nagappan, N., Maximilien, E. M., Bhat, T., & Williams, L. (2008). Realizing quality improvement through test driven development: results and experiences of four industrial teams. Retrieved April 12, 2016, from http://research.microsoft.com/en-us/groups/ese/nagappan_tdd.pdf

Orso, A., & Rothermel. (2014). Software Testing: A Research Travelogue (2014). Retrieved March 31, 2016, from http://www.cc.gatech.edu/~orso/papers/orso.rothermel.ICSE2014-FOSE.pdf

Sanchez, J., Williams, L., & Miximilien, E. M. (2007). A Longitudinal Study of the Use of a Test-Driven Development Practice in Industry. Retrieved March 31, 2016, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.6319&rep=rep1&type=pdf

Walther, S. (2009). TDD Tests are not Unit Tests | Stephen Walther. Retrieved April 8, 2016, from http://stephenwalther.com/archive/2009/04/11/tdd-tests-are-not-unit-tests

Williams, L., & George, B. (2004). A Structured Experiment of Test-Driven Development. Retrieved April 11, 2016, from https://pdfs.semanticscholar.org/616d/3f7e831c725b51220a34fbee3ca6ac1d711c.pdf

Williams, L., Kudrjavets, G., & Nagappan, N. (2009). Unit_testing_cameraReady.pdf. Retrieved March 31, 2016, from http://collaboration.csc.ncsu.edu/laurie/Papers/Unit_testing_cameraReady.pdf

Williams, L., Maximilien, E. M., & Vouk. (2003). Test-Driven Development as a Defect-Reduction Practice. Retrieved April 11, 2016, from http://collaboration.csc.ncsu.edu/laurie/Papers/williamsltestDrivenDevelopment.pdf

Appendices

Appendix A – Interview Introduction

Introduction

My name is Raquelita Rós Aguilar and I am starting my BSc. research study as my final project. In this study I am doing a research to investigate the effectiveness of using the method Test-Driven development in software projects.

I would like to interview 8 active developers that is using or has used TDD in a project with significant results and compare that to a non-TDD project developed by the same person. By comparing these two projects I'll be getting a better insight and understanding on how and where TDD affects the overall outcome of a project. I will summarize the results from the interviews and hopefully find some answers to my questions. For me to get the most out of this interview, I request to be allowed to record it.

> *Notice that this study will be completely confidential and anonymous. Your name or the company's name will not be used at any time during this research. After processing the data from this interview I will terminate them completely.*

I want to thank you in advance for taking the time to assist me with this study and letting me interview you. Your time is valuable to me and I predict that this interview will take no longer than one hour of your time.

Kind regards,

Raquelita Rós Aguilar
Mail: raquelita15@ru.is
Phone: 660-6089

Appendix B – Background

1. What does your company develop?
2. How long have you been in software developement?
    a. Experience level:
        i. > 15 years
        ii. 11 – 15 years
        iii. 6-10 years
        iv. < 5 years
3. How many people work in your company as developers?
4. What programming language do you mainly use?
5. When did you first hear about TDD?
6. What was your first impression of this approach?
7. What is your impression after using it for some time?

Appendix C – TDD Background

1. How long have you been using TDD?
2. Where did you learn using TDD?
3. Did you find it hard to learn TDD?
4. How do you know if you are writing a good or bad unit test?
5. Do you think TDD is time consuming, that is, would you rather spend your time on something else?
6. In your opinion, what do you think the advantages are?
7. What do you think the disadvantages are?
8. Do you use TDD in all of your projects?
    a. If not, what type of projects do you use TDD in and what type of projects do you not use them in?
    b. Why do you only use TDD in those projects?

Appendix D – TDD Project

1. What is the TDD project description?
2. How many people were on the team?
    a. What was your role on the team?
3. How big was this project and how long dit it take to develop it?
4. What programming language was used?
5. Why did the company / team decide to adopt TDD at that time?
6. What was the problem you were facing and how did TDD solve that problem, if it did?
7. Did you follow any specific development methods, such as Kanban, Scrum, etc?
8. What were the advantages of using TDD in this project?
    a. Improved Code Quality?
    b. Improved product / software?
    c. Improved user experience?
    d. More productivity with the developers?
    e. Reduced defects?
    a. Reduced maintenance after delivery
    b. Savings for the customer?
    c. What is your understanding of why these advantages took place?
9. What were the disadvantages of using TDD in this project?
    a. What is your understanding of why these disadvantages took place?

Appendix F – Non-TDD Project

1. What is the project description?
2. How many people were on the team?
    a. What was your role on the team?
3. How big was this project and how long dit it take to develop it?
4. What programming language was used?
5. Did you have any issues / problems through out the implementation and / or delivery?
6. Did you follow any specific development methods, such as Kanban, Scrum, etc?
7. Did you use any formal methods for testing through out the implementation?
8. Do you think that there were any advantages of not using TDD in this project?

    a. What is your understanding of why these advantages took place?
9. Do you think that there were any disadvantages of not using TDD in this project?
    a. What is your understanding of why these disadvantages took place?
10. What is the reason that TDD was not used?

Appendix E – Closing Questions

1. Do you recommend TDD for others to use in the future?
2. Do you think there should be put more emphasis on teaching TDD in universities?
    a. Why do you think that?
3. What do you think the ruling differences are between TDD and non-TDD projects?
4. Is there something that you would like add?