A case study of Test Driven Development

Thesis · August 2017

DOI: 10.13140/RG.2.2.27852.92803

1 author:

Meya Stephen Kenigbolo
University of Tartu
1 PUBLICATION **0** CITATIONS

SEE PROFILE

UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Meya Stephen Kenigbolo

A Case study of Test-Driven Development

Master's thesis (30 ECTS)

Supervisor: Dietmar Alfred Paul Kurt Pfahl
Co-supervisor: Kaarel Kotkas

Tartu 2017

A Case Study of Test-Driven Development

Abstract:

The purpose of this study is to analyse the benefits and/or drawbacks regarding the implementation of Test Driven Development (TDD) as part of the software development lifecycle of startup companies. This study was conducted in three phases: The first phase focused on a study of the current TDD implementations in an early stage startup company assigned the task of delivering Software as a Service(SaaS) product to their clients (Company A). The main purpose of this stage will be to analyse the current existing software development methodology and what role (if any) TDD plays in the entire process. The second phase revolved around identifying the current existing practices of TDD in a company that has successfully embedded the practice into their software development lifecycle (Company B). This phase involved an in depth analysis of the TDD practice in Company B: how it was first introduced, the challenges faced during the initial stages of implementation, reasons for its adoption as well as their views on the future of TDD. The third and final phase focused on gathering data from other companies that practice TDD and how the knowledge acquired from this study can be used to make a data driven decision regarding the benefits/drawback of TDD for company A.

CERCS: P170

Keywords: Test Driven Development, TDD, Testing, Test First Development, Software Development Lifecycle

Testipõhise arenduse juhtumiuuring

Lühikokkuvõte:

Magistritöö eesmärk on analüüsida idufirmade näitel test-driven development (TDD) tarkvara arendusprotsessides rakendamise tugevusi ja nõrkusi. Magistritöö kirjeldab uurimust kolmes etapis: esimene etapp keskendub varajases staadiumis idufirmade uurimisele, kus on juba TDD rakendatud ning kus firma ülesandeks on rakendada tarkvara teenuse (ingl. k. SaaS) toodet oma klientidele (firma A). Selle etapi eesmärgiks on analüüsida hetkel olemasolevat tarkvaraarenduse metoodikat ja millist rolli täidab TDD kogu protsessis. Teine etapp keskendub hetkel kasutatavate TDD praktikate tuvastamisele ettevõttes, mis on edukalt juurutanud nimetatud praktika oma tarkvaraarendusse (firma B). See etapp koosneb põhjalikust TDD praktika analüüsist firmas B - kuidas võeti TDD esmakordselt kasutusele, juurutamisel esinevad väljakutsed, TDD kasutuselevõtmise põhjused ning idufirma nägemus TDD tuleviku suhtes. Kolmas ehk viimane etapp keskendub andmete kogumisele teistelt ettevõtetelt, mis kasutavad TDD-d ning analüüs, kuidas saaks antud uuringust saadud andmepõhiseid teadmisi kasutada otsuse langetamiseks firma A jaoks, arvestades TDD eeliseid ja puudusi.

CERCS: P170

Võtmesõnad: testidel põhinev arendus, TDD, testimine, testimise esimene arendus, tarkvaraarenduse elutsükkel

Table of Contents

1. Introduction

In recent times the concept of Test Driven Development has evolved rapidly that there is the tendency to totally forget that one of its central components is Test First Development (TFD)[26][14]. TFD is generally seen as an umbrella term for several different approaches which involve the art of writing tests before actually writing code itself. TDD can be interpreted in different ways and does have several definitions to different people, however, for the purpose of this study I will be considering TDD to be a type of software development process which involves writing a single test (as opposed to specifying a single behaviour), watching this test fail and then going ahead to write just enough production code that will ultimately make the test pass[17][8][10][16][11]. A lot of development teams refer to this as the process of making the tests go green from red by first writing the test. According to Microsoft researchers[1] in the paper "Empirical Software Engineering at Microsoft Research" they described TDD as the art of writing failing unit tests and then going on to write implementation code to that will make the failing tests to pass. In researching a number of scientific papers which have been published regarding TDD one cannot necessarily make a case for the implementation of TDD as an Agile software development methodology without having access to some form of data for which further analysis could be conducted. For this particular reason we will be conducting this case study by interviewing both Company A and B as well as reviewing several literatures on Test Driven Development methodology and implementation to provide useful information on the benefits and/or drawbacks that TDD implementation will bring to Company A. This will be achieved by both quantitative and qualitative analysis of the relevant information.

The precise research questions for this study are as follows:

- RQ1: What is the current state of TDD in both companies (A and B)?
- RQ2: What are the expectations of TDD in both companies (A and B)?
- RQ3: How has TDD helped company B?
- RQ4: What are the success factors for introducing TDD according to the literature?
- RQ5: How can the results from both the literature study and case studies be used to facilitate the introduction of TDD in company A?

These research questions will enable us propose a method for the adoption and implementation of TDD in the startup company.
The thesis itself will consist of eight chapters after a detailed abstract of the case study. Chapter two gives an overview of the current state of the art regarding TDD. Chapter three will cover the background of the study which will then proceed to Chapter four where I discuss the research methodology used for the literature survey and case study design. Chapter five will be focused on the results from the literature survey and crucially attempt to

answer RQ2, RQ4, and RQ5. Chapter six will analyse the results from interviews and questionnaire data collected from both companies. This should provide answers to RQ1, RQ3 and RQ5. The penultimate chapter seven covers the recommended solution for the adoption of TDD in a startup company. Chapter eight will give the conclusion of the case study and the document ends with a list of references in chapter nine and an appendix.

2. Review of state of the art

A couple of articles on the history of TDD[15] document that TDD had been in practice even before the computing era, the most notable being Ada Lovelace Byron[15]. During the early days of computing (the mainframe era) TDD was also an engineering practice being used, however, the modern state of TDD is widely attributed to Kent Beck (often referred to as the inventor of Extreme programming) who in 1994 wrote first version of SUnit test framework. TDD became largely acceptable in the software development community mostly due to the Agile Software and Extreme Programming movements.

2.1 History and Evolution of TDD

Modern TDD differs from TDD in the early computing era and the most prominent difference can be attributed to the testing paradigms. While early TDD was in most cases an implementation of manual testing, modern TDD was simplified via automated testing[14].

Although the SUnit suite written by Kent Beck is still referred to as the first modern TDD framework, modern TDD came to life with the JUnit tool. On the 16th of August 2000 the website JUnit.org was launched and this was quickly followed by the NUnit framework which was registered on sourceforge on the 2nd of September 2000 and about two months later JavaUnit was also registered on sourceforge (25th November 2000).

JUnit was well received by Java developers who were practicing one of Agile or Extreme Programming as their prefered software development methodology. In recent times, several unit testing frameworks whose structures are modeled off Kent Beck's SUnit has emerged. Collectively they are referred to as the xUnit class of tools and are available for almost all modern programming languages e.g CUnit for C programming language, CppUnit for C++ programming language, RUnit for R programming language etc.

2.2 TDD Implementation

It is imperative to understand what exactly test driven development is and how it is actually implemented. TDD follows four simple steps

1. Write a test that fails: This implies that before you actually do write down any line of code for a specific functionality, e.g. a method to calculate tax, you begin by writing a test for the functionality of the said method and also the minimal amount of code

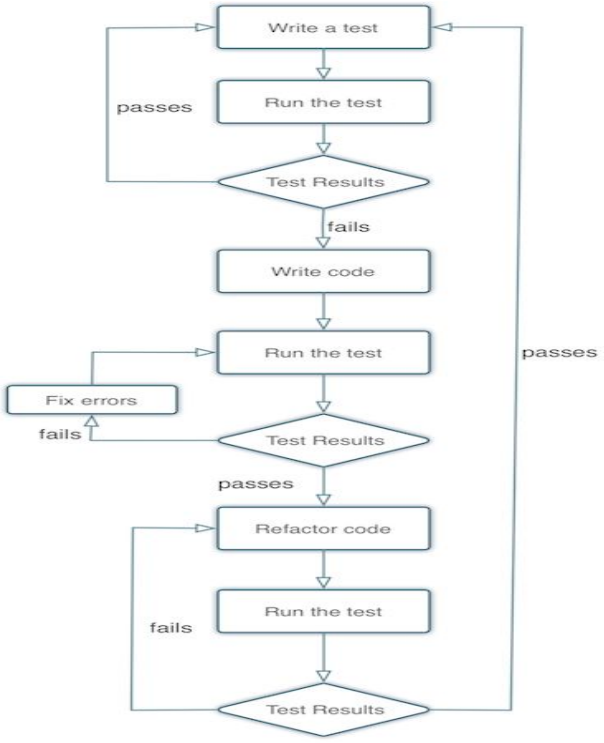
which will be required to enable the test actually run e.g. method definition. This is the first and most important step in TDD as it is how every test should begin[4].

2. Write code to enable the test pass: In this step the developer proceeds to implement the functionality that will enable the test to pass. While writing the code it is often the case that the test is run to see what part of the code is functioning. This immediate feedback received in real time is mostly believed by those who practice TDD to actually improve the developer's productivity[4].
3. Refactor the code (Optional): In most cases this is said to be a step in TDD performed after the test passes. Refactoring makes the code more concise and precise[4].
4. Repeat: This requires the programmer/developer to repeat the previous steps for every functionality they wish to implement[4].

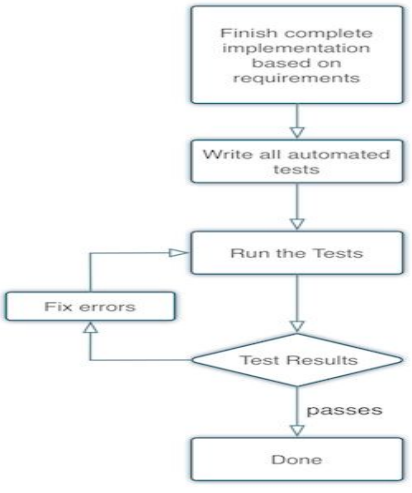
2.3 Difference between Traditional testing and TDD

The most significant difference between traditional software testing and TDD is the test first factor[5]. The traditional software testing paradigm follows the test last approach where the entire code implementation is written out before any tests hence testing is seen primarily as a verification mechanism to ensure that the implementation functions as it was intended to.

The chart below shows the difference between traditional testing cycle and that of TDD.



TDD cycle



Traditional Cycle

Fig 1 Difference between TDD and Traditional Software Development cycles[14]

2.4 Summary

TDD as a software engineering methodology is built upon extreme programming and has evolved rapidly in the last decade. It emphasizes a test first approach and this is exactly how it differs from the traditional software development cycle.

3. Research Methodology

In this section I will give an overview of the research methodology. A mixed research [27] shall be adopted to enable me find the appropriate answers to the research questions posed in chapter one.

RQ4 and **RQ5** will be answered based on a systematic literature review. In order to perform this literature review I will be following the guidelines to perform systematic literature reviews as proposed by Kitchenham et al[27]. Although the review of the state of the art starts with some form of literature review it is of little of no scientific value to the research goals of this study.

For this study Kitchenhams literature review protocol has been prefered over other literature review protocol purely based on the fact that it aims to present a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology[27]. This enables a researcher to drive conclusions based on the data gathered. The results from this literature survey will be analysed and reported accordingly

To enable me find the answers to **RQ1**, **RQ2** and **RQ3**, a case study will be designed. This case study will comprise of a survey(questionnaire) and interviews. The main motivation behind this case study will be to gather data at its point of origination in order to acquire accurate data for analysis and results reporting. The case study will be conducted according to Runeson, P. & Höst [30] laid down guidelines on conducting and reporting case study research in Software Engineering.

The data which will be both quantitative and qualitative in nature will be analysed and reported according to the given guidelines. The oral interviews which result in qualitative data will be analysed first by transcribing the interviews into written form and then analysed using thematic analysis. Using coding the transcribed interview will be segmented by different parts and phrases as it relates to the research questions.

The respondents for the case study questionnaire and interviews will be drawn from the two major case study participating companies as well a few respondents from the industry who currently practice TDD.

4. Literature Review

In this section we will review studies related to TDD including but not limited to studies relating to it's origin, it's practices as well as the effects of TDD in productivity.

This chapter gives an overview of exactly how the relevant scientific articles relating to this case study were found and how the information acquired was used to determine how the respondents were selected and interviews set up. The results from the literature survey will be used as the guideline for the case study questionnaire design.

4.1 Literature Survey Design

The literature survey design follows Kitchenham et al[]. In order to bring into play an evidence based way/style in Software Engineering practice, Kitchenham suggests that researchers in the Software Engineering discipline should adopt 'Evidence Based Software Engineering. For this study, evidence is said to be the unification of software engineering studies or a high quality that relate to the research questions formulated in chapter 1.

The incentive behind carrying out a literature survey in this study is to enable me identify and analyse the effectiveness and benefits as well as drawbacks and/or factors limiting the adoption of Test Driven Development.

4.1.1 Review Protocol

The essence of this review protocol is to clearly specify the process which the literature review follows in order to identify and collect evidence based on several sources including journal articles, scientific research papers and data that are related to TDD. By presenting the review protocol I hope to make visible the motivation behind the selection strategy especially for a situation where there might be misunderstandings towards selection and inclusion of studies.

4.1.1.1 Objectives

The main goals which are responsible for conducting this review are as follows

1. Identify and analyse the effectiveness and benefits of Test Driven Development
2. Describe the factors limiting the adoption of Test Driven Development.

4.1.1.2 Inclusion / Exclusion Criteria

Inclusion:

1. Primary focus on journal, conference and workshop articles
2. Date of publishing not earlier than January 2000

3. Availability of the journal, conference or workshop articles on Google Scholar via the University of Tartu Library interface.
4. Studies that involve both professionals and students from the industry and academia respectively.
5. The study is available in full text mode.
6. The study reports factors that limit TDD adoption
7. The study follows a defined literature review guideline e.g. Kitchenham

Exclusion:

1. Articles which are not in English.
2. The abstract or conclusion not making any defined reference to Test Driven Development experiments either academic or industrial.
3. Studies that do not contain any form of reported quantitative and/or qualitative evidence whatsoever in regards to TDD.
4. Journals, articles, conference papers which are not in the field of Computer Science and/or Software Engineering.
5. Journals, articles, conference papers with less than 10 citations.
6. In-print journals, articles and conference papers that I do not have access to

4.1.2 Search Strategy

The primary databases used for searching relevant literature are Google Scholar, SpringerLink and ACM digital library however I also did search the keywords mentioned above on Gdinwiddie biblio [3] although the query produced largely the same results as the other databases.

Data Sources

The databases the searches were conducted on are as follows;

1. ACM Digital Library
2. Google Scholar
3. SpringerLink - Computer Science
4. Gdinwiddie Biblio
5. Science Direct

Review Method

The retrieved studies from the search following a detailed screening. I used a five step analogy in order to remove duplicates of articles and sort out relevant articles to be reviewed for this study

1. Removed studies based on duplicate titles
2. Sort out relevant studies based on their title, abstract and keywords
3. Selecting studies (articles, journals) based on detailed review/screening
4. Citation count

For the purpose of this case study, I made use of the Google Scholar database and SpringerLink – Computer Science in searching for relevant literature by using the search terms "TDD" and "Test Driven Development". I proceeded further to make use of the ACM digital library and tailored my search towards articles written on or after January 2010 with the above mentioned keywords. As TDD is related to the core concept (test-first) of extreme programming which began in the late 90's[5]. This yielded a total result of 540 articles.

In order to access the relevance of the papers, I filtered to return only articles with more than 10 citations and this resulted in a total of 42 articles for review. I read through the abstracts and conclusions as well as the references to ascertain if they were relevant for this thesis. During this process, I was able to extract details of other articles as well as make an informed decision regarding the relevance of each read literature to this study. One study which I did find useful although didn't meet the citation criteria was also added[37]. After all exclusions of duplicates and relevance of articles decision I ended up with a total of 34 articles, journals and conference papers for the literature survey

4.1.3 Extracting relevant information from literature

A total of 34 articles, journals and conference papers were selected for this literature study, whilst reading through I noted down pages and sub headings which I believed were relevant to the thesis and appropriately marked the research question in which they correlate directly to. This resulted in two main research questions (**RQ4** and **RQ5**) being correlated to the pages and sub headings I noted down.

From the different journal articles I read, I gathered that empirical investigations (e.g. experiments) into the effects of TDD can be influenced by different variables e.g. academic vs industrial experimental settings, developers skill et al.

4.1.3.1 Existing Research on TDD

According to Microsoft researchers[1] in the paper "Empirical Software Engineering at Microsoft Research" they described TDD as the art of writing failing unit tests and then going on to write implementation code to that will make the failing tests to pass. This

description seems to be the widely acceptable definition of TDD as echoed by H Erdogmus in his paper "Effectiveness of Test-first Approach to Programming",[4] as well as Nachiappan Nagappan in his 2008 publication[6] and several other research articles on TDD also agrees and asserts this definition[16][11][8].

Several individuals and groups/researchers since the early 2000's when TDD became quite famous, have tried to evaluate the concepts of TDD and how effective it is or can be[6][9][10][11][12]. There have been varying results on the strengths and weaknesses of TDD which at most times have been similar although in few cases, the results have been very different. Whilst this thesis is focused on analysing TDD from a different end of the spectrum, there's a lot or relevance that can be achieved from the conclusion of these studies.

4.1.3.2 The IBM RSS Case Study

IBM had a software development group focused on the IBM retail store and they built a non-trivial software system which was based on a stable standard implementation of TDD. By adopting the TDD approach they ended up in reducing their defect rate by approximately 50% in comparison to another similar system (one which they developed using an improvised unit testing approach)[2]. At the end of the project it was also discovered that the project was completed on time and the implementation of TDD had only a minimal development productivity impact[2].

Prior to TDD implementation, ad-hoc unit testing was used. This type of testing required the developer to write the code for the important classes and then create a UML class and sequence diagram. This unit testing approach was a post coding activity relying on different methods of unit testing were undisciplined and were more or less done as an afterthought.
From IBM's experience with the ad hoc testing they discovered that the approach in almost all cases led to last minute testing and in some cases no form of testing at all.

Study Conclusions

The results of implementing TDD for them were as follows;

1. 50% improvement in the defect rate of their system.
2. They had an automated unit test coverage of the developed classes at 86% which was higher than their initial target of 80%.
3. The final product was more accommodating to future evolution and late changes.
4. Using daily integration saved them from late integration problems as there was always a functioning part that had been tested

Their feedback for teams who are looking towards transitioning to Test Driven Development include the following key points;

1. Apply TDD from the inception of the software project
2. Usher in automated build test integration when approaching the second part of the development phase for the project[12].

4.1.3.3 The Microsoft Case Study[1]

In 2006 researchers at Microsoft undertook a case study on the effectiveness of Test-Driven Development in a corporate, professional environment[1]. This research was conducted by two Microsoft researchers namely Thirumalesh Bhat and Nachiappan Nagappan. Their research was mainly based on the effects of TDD in defect reduction. Two projects from Microsoft were chosen by the researchers, one which was part of Windows and the other part of MSN. To determine the effects of TDD in defect resolution they did an in depth analysis of the bug tracking system as well as keeping track of the time it took for the software engineers who worked on the product to actually complete their tasks. For each of the two projects the researchers selected a group of similar projects in size and them via their issue tracker.

For the first case study which deals with code that is a part of the windows operating system, the first team was composed of six developers and they were able to write 6,000 lines of code over a period of 24 man-months using TDD and this was compared with another windows project in which two two developers wrote 4,500 lines of code in 12 man-months without using TDD. The researchers found a reasonable amount of improvement in code quality when Test Driven Development was implemented.

Study Conclusion
The development team which did not implement Test driven development produced 2.6 times as many defects as the group which implemented TDD did. Also in the second case study conducted, the non-TDD group produced 4.2 times more defects than the team which implemented it.

4.1.3.4 Industrial context research

From the literature survey I discovered that most of the evidence reported in regards to TDD heavily focused on the following:

1. Internal code quality
2. Productivity
3. External code quality

Thirumalesh Bhat and Nachiappan Nagappan reported a significant increase in code quality[1]. In almost all case studies when time factor was taken into consideration, TDD

required more development time[1][2][10][11][12]. Also, in another experiment carried out by George and Williams[11] it was discovered to increase development time by 16% however when the comparison of TDD with the waterfall development approach was conducted it was discovered that TDD developers passed 18% more functional black box test cases than their waterfall counterparts.

Thirumalesh Bhat and Nachiappan Nagappan reported a 15% - 35% increase in development time[1]. However, in George and Williams experiment[10][11], they found that TDD did improve the productivity and effectiveness of the experiments subjects, as well as to lead to high test coverage. The results and experiences of the research conducted by four industrial teams as observed by Thirumalesh Bhat, Nachiappan Nagappan, Maximilien, L.Williams[6] summarized the microsoft and IBM studies, indicated that there was a decrease in the pre-release defect density of the four projects observed to the tune of about 40% - 90% when compared with similar projects that didn't use the TDD practice.

In the study conducted by Visaggio et al.[33], a controlled experiment with industry professionals was performed. The focus of this experiment was to identify if TDD plays any role in improving unit tests. The results from their experiments reports that TDD does improve the overall unit testing although this was at the expense of the development process which became slow as a result.

Abrahamsson et al.[36] in their study conducted on a team at Nokia Siemens over a period of three years reported that TDD significantly improved the quality of their code as well as simplifies the maintenance process for their software. Based on additional interviews they report no negative effects regarding the application of TDD for the long term period.

Table 1 below lists the findings of research studies performed in industrial settings.

Study	Point of comparison	No of Subjects	Quality effect	Productivity effect	Comments
George and L. Williams [10][11]	Controlled experiment vs TLD	24	18% more test cases passed[10]	TDD resulted in about 16% extra time	98 % method, 92 % statement and 97 % branch coverage with TDD

Maximillien and Williams[2]	Case Study vs Adhoc unit testing	9	50% reduction in defect rate[2]	Minimal effect in productivity	Automated test coverage of 86%[2]
Thirumalesh Bhat, Nachiappan Nagappan, Maximilien, L.Williams[1][6]	Case Study	4 projects	pre-release defect reduced by about 40% - 90%	15%- 35% increase in development time	Teams which didn't use TDD had a higher defect rate
Visaggio et al.[33]	Controlled experiment vs TLD		No difference	No difference	This experiment considered unit test quality and productivity as the metrics it aimed to measure. It reported no effects on both metrics by TDD
Madeyski et al[37]	Controlled experiment vs TLD		No difference	Higher productivity when using TDD	In every case subject productivity increased although this wasn't

					significant
Natalia Juristo et al[17]	quasi-experiment	30	No significant statistical change	No significant statistical change	The results show that TDD skills' set is a factor that could cause up to 28% of the external quality, and up to 38% for productivity.[17]
Williams et al[12]	Case Study		Slight decrease in developers productivity	More time spent writing test cases	The defect rate was significantly better for the new system when compared with the legacy system

Table 1 - Industrial case studies

4.1.3.5 Academic context research

TDD research in the academic context have yielded quite different results most times from those conducted in the industrial context. In the study conducted by Mueller and Tichy [20] where they examined different Agile software development methodologies, including TDD, within a university course, they discovered that TDD was seen as a difficult and hectic software development methodology to adopt because writing tests before actually writing code seemed almost impossible to do. However, Gupta and Jalote [21] reported that students felt quite confident regarding testing effort applied by using TDD, believing that it would actually yield better results than the traditional Test Last Development approach.

Pancur et al.[22] study reported that students perceived TDD as way more difficult for professionals to adopt. They believed practicing TDD will hinder their productivity, efficiency, and the quality of their code. Erdogmus, Morisio, et al. 2005 study[4] found no real difference in the quality of the code when TDD was used although it reported to have more quality results. Despite the fact that they reported increased productivity, there were however no specific figures given. In general it did seem to suggest that developer testing practiced within Extreme Programming was indeed useful. Natalia Juristo et al.[8] in a study conducted at the University of Basilicata (Italy) reported that TDD does not affect testing effort, software external quality, and developers' productivity. Pancur and Mojca[34] in their study reported that the benefits of TDD are small when compared to TLD. The positive results were in terms of code quality and productivity.

Muller et al.[31] administered an experiment using computer science students as the subjects for comparison between Test Driven Development and Test Last Development. The corresponding results showed that neither TDD nor TLD accelerated the implementation process or made the resulting programs necessarily more readable although TDD did show slight evidence of more readable and more maintainable code to it's TLD counterpart. Huang and Holcombe[32] in their experiment reported that the group which used TDD passed more acceptance tests than the TLD group however it is worth pointing out that this trend is as a result of time taken on unit tests rather than the TDD methodology.

Study	Type/Point of comparison	No of Subjects	Quality effect	Productivity effect	Comments
Erdogmus, Morisio, et al. 2005[4]	Controlled experiment/ Iterative test last	24	No difference	TDD more productive	More consistent quality results with TDD
Pancur, Ciglaric, et al. 2003[22]	Controlled experiment/ Iterative test last	38	No difference	No difference	Student believe TDD isn't effective

Davide Fucci, Burak Turhan et al[8]	Controlled experiment/ Iterative test last	58	No difference	No difference	TDD doesn't affect productivity, software external quality or testing effort
Haung et al[32]	Controlled experiment/ Iterative test last	39(2006) 60(2004) 80(2003) 96(2002)	Little or no difference	70% higher	Productivity in Test first team was 70% higher although there was little or no difference in software quality despite the increasing test effort
M. Muller et al[31]	Controlled experiment/ Iterative test last		No difference	No difference	TDD neither improved code quality or productivity
Gupta et al[21]	Controlled experiment/ Iterative test last		Significant improvements in code quality	No difference	This study reported improvements in the code quality

Table 2 - Academic case studies

5. Case Study Design

The case study is designed according to the guidelines for case study research in software engineering as postulated by Runeson, P. & Höst, M.[30]. The purpose of the case study is to gather information from the participating companies in order to answer **RQ1**, **RQ2** and **RQ3**. The overview of the case study design is as follows:

5.1 Selection of participating companies

In order to select the participating company, I took into consideration the following factors

1. Located in the EU/EEA
2. Relationship to the startup ecosphere.
3. Startup companies which have been functioning for at least two years.
4. Companies focused on SaaS application(s)
5. Not a one-person startup company (Minimum of 3 developers).

According to an article published on Zdnet[19], Poland is the number one destination for software outsourcing in Eastern Europe hence I compiled a list of International outsourcing companies in Poland which focus on software development for a more established company to use as Company B in the case study. I tailored the search towards outsourcing companies that use a more modern high level programming language such as Ruby or Python for software development.

Company A

This is a start-up company located in Estonia. It focuses on the development of a SaaS application. The company is young, made up of a small development team responsible for building the core of the application. The company has an approximate size of 10 persons with about 4-5 developers and they are enthusiastic about the possible benefits/drawbacks the implementation of TDD will bring to the company.

Company B

An international company located in Poland has been practicing TDD for roughly 5 years. The company is approximately 7 years old and is a software development company, which develops and maintains several SaaS applications for their client base. They have separate teams including Marketing, design, frontend, backend, QA etc. and each team consists of approximately 7 – 10 persons. Their expertise in TDD and close relationship to the start-up ecosphere (having being one themselves as well as building SaaS products for startups) made them a good choice for the case study

Other Respondents

Using the above mentioned selection criteria for selecting companies, survey(questionnaires) were sent out to several companies matching the criteria for voluntary responses. Apart from one respondent company which is domiciled in Bulgaria, the rest of the companies which responded to the questionnaire were all based in Estonia.

5.2 Selection of company respondents

In order to get useful information that will be of benefit to the case study, we had to establish the criteria for selecting respondents for the case study. In a hierarchical manner from most important:

1. Most senior developer or QA (in terms of time spent at the company as opposed to experience level)
2. Developers or QA who has been with the company for a minimum of 10 months

This restricted the respondents to 4 in company A and 12 in company B. All four in company A were willing to participate however only ten in company B were available within the timeframe of this study as two of company B's respondents were going to be away on vacation.

5.3 The Interview Process

The structure of the interview for this case study was as follows:

1. I began by getting Company A acquainted with my research topic and the formal interview process explaining the purpose of the interview in person. For company B the same was done however this was first carried out via email communication before being done onsite in person.
2. I gave a detailed exposition of the key points noted from the literature survey, ensuring that I explained the variables involved most especially the academic vs industry related experiments.

The proposed time allocated to each interview was 20 minutes while the time estimated to finish filling out the questionnaire was 10 minutes. These are proposed times and could differ in some cases.

5.4 Data Collection Process

In order to allow the companies allocate time for the interviewees to respond to the interviews and questionnaires a data collection schedule was proposed to span a week-long (working days) duration. Questionnaire data will be made available to respondents using an online survey service (Surveymonkey) [18] that enables the responses to updated real time after submission by the respondent. Fig 2 below is a table showing the proposed one-week

structure for company A and B. Subsequent schedules (formal/informal) will be made as an addendum in the appendix.

S/No	Max Interview duration (In minutes)	Company A	Company B
1	20	1	2
2	20	1	2
3	20	*	2
4	20	1	2 *
5	20	1	2

*extra time available

Table 3 - Schedule for Interview

5.5 Results from the Case Study

Below we analyse the results of the case study (questionnaire and interviews) as it relates to the different research questions to be answered by the case study.

1. **RQ1**: What is the current state of TDD in both companies (A and B)?
2. **RQ2**: What are the expectations of TDD in both companies (A and B)?
3. **RQ3**: How has TDD helped company B?

The answers to these questions are collated through analysis of the questionnaire responses and thematic analysis on the interview data. Thematic analysis was used for the interviews due to the fact that unlike the questionnaire which had quantitative data, the information from the interviews were qualitative.

To enable proper analysis of the interviews I began by first listening to the interviews after which I proceeded to transcribe them. After the transcriptions I proceeded to segmenting different parts and phrases of the transcribed interviews as it relates to the research questions

and this helped me to remove unnecessary repetitions from the transcription as all key points could be matched to a theme that answers one of the three research questions above. In some interviews where I didn't have access to playback I engaged in actively writing down summaries of important points highlighted and this was also used in the analysis.

5.5.1 Questionnaire answers and interview summary relating to RQ1

The answers to questionnaire questions related to this research question can be found in the tables below. The question number corresponds to that of the number in the questionnaire. The question is listed as well as the different answers from the respondents.

Question No	Question	Company	No. of Respondents
2	**How long have you worked at the company?**	A	4
		B	10
		Other Respondents	8

Table 4 - Summary of Question 2

Company	Less than 10 months	10 months - 2 years	2 years - 3years	Over three years
A	1	3		
B		4	3	3
Other Respondents		3	3	2

Table 5 - Responses to Question 2

Question No	Question	Company	No. of Respondents
4	**How many of the developers in your company regularly use TDD in their projects?**	A	4
		B	10
		Other Respondents	8

Table 6 - Summary of Question 4

Company	Nobody Applies TDD	Some developers apply TDD regularly	About half of the developers use TDD regularly	Most developers apply TDD regularly	Everybody applies TDD
A		3	1		
B	1	2	2	3	2
Other Respondents		1	2	3	2

Table 7 - Responses to Question 4

Question No	Question	Company	No. of Respondents
5	How long have you practiced TDD?	A	4
		B	9
		Other Respondents	8

Table 8 - Summary of Question 5

Company	Upto 1 year	1 - 2 years	Over 2 years
A	3	1	
B	3	2	4
Other Respondents	2	3	3

Table 9 - Responses to Question 5

Question No	Question	Company	No. of Respondents
3	What is your job role?	A	4
		B	10
		Other Respondents	8

10	**Briefly describe your company TDD cycle**	A	4
		B	10
		Other Respondents	8

Table 10 - Summary of Question 3 and 10

Company	**Job Role**	**TDD Cycle**
A	Developer	It's very basic and voluntarily. I like it especially when designing bigger API's and start from top to bottom TDD. We don't have any rules about it though :)
	Developer	There isn’t a defined TDD process
	Developer	I am not really sure if what we practice is TDD as sometimes we write tests before the code but in most occasions the tests come after we are through with writing code
	Developer/Project Manager	We try when possible (if we have the time) to write tests for certain use cases and edge cases before actually implementing it via code
B	Developer	The TDD cycle is already defined by the QA and team lead. All we do is to follow their lead by writing the tests before writing the implementation code. I cannot say for sure if this process is strictly followed by everyone but this is the laid down procedure and I try to follow it
	DevOps	I do not practice TDD in the context of my work however there is a laid down TDD principle which the QA team expects developers to adhere to although how much this is strictly followed I cannot say for sure. Whenever I do join in on feature development I try to follow the process to the later
	Developer	First is the card definition, there should be no blank spaces for the developer, he/she should not be making the business decisions. Afterwards the card is split into as small chunks as possible. Next the development process starts.
	QA	For new functionality: 1. Write a unit test with expected outcome, based on the requirements for a given feature. 2. Write a piece of code. 3. If it passes, write a new test or expand the existing one.

		4. Repeat from step 2. For changing existing functionality 1. Cover functionality with tests, if some are missing. 2. Write tests for the behaviour. 3. Adjust code to pass the new test case and do not break the old ones.
	Developer	I follow the style given by the QA guys as much as possible
	Developer	Write tests and the write the code that will allow the tests to pass. Seems quite simple at first but is easier said than done.
	Developer	Only 1 developer form our company does TDD in it's entirety I think. The rest of us mostly just focus on writing the tests to cover the use cases and try to commit it before we commit the main code. This doesn't mean that we follow the principles strictly does it?
	IT Project Manager	QA team defines this
	Developer	Implement TDD for the most important parts based on priority and afterwards If there is time some people do it for the other parts as well.
	Team Lead (QA or Software Development)	Since most of the time tasks are under time-pressure and delivery is needed fast, developer needs to analyze, what part of code is more important to be tested and which part is not. Mostly it happens that some of task is being done TDD and other parts with tests afterwards. Because of this, we cannot reap full benefits from TDD since we can do it occasionally.
Other Respondents	Developer	Implement TDD by following the cycle of writing tests and then writing the code to make the test pass and afterwards refactor if necessary. Most times refactoring makes the code more diffused and sometimes I end up making it more complex so because of this I don't refactor too often
	Developer	Write test, Write implementation code, refactor and then repeat the process again
	QA	Test - Implementation Code - Refactor - Test. For those of us in QA we simply do A/B testing however in sometimes we are required to write test cases even before the product is designed and the developers have to write the code that makes it run. The advantage for us is that we write the test in phases of the development process. I'm not sure if this qualifies as TDD exactly for the developers because my understanding of TDD is

		that the developers write the test cases themselves. I will like to know what your research thinks about this if it is something that you cover in this research.
	Senior Engineer	Write failing tests - Write small enough code to make the test pass, refactor code and repeat the cycle. Sometimes this is not entirely the case though depending on the urgency of the needed feature i.e if it's a new feature or a feature advancement.
	Software Engineer	Tests first (Red) and then code for the test to pass(Green) and then refactor(I think this is usually denoted with the yellow right?) and finally continue just like you started.
	Developer	Not entirely sure what parts of the process you request as it seems some carry a higher priority than others but generally we use the test first then write the code to make the test pass type of TDD
	Head of IT	We outline our procedure following the red to green to yellow style by writing the tests first and foremost before writing any sort of implementation code. This is something we try to instill in all our developers
	CTO	Our team is a small team and I cannot say we practice TDD to its entirety but whenever we do we follow the best practice of TDD which is writing the tests before actually coding. Depending on the time available we sometimes have skipped using TDD

Table 11 - Responses to Question 3 and 10

Question No	Question	Company	No. of Respondents
6	**Did you have TDD experience prior to joining the company?**	A	4
		B	10
		Other Respondents	8

Table 12 - Summary of Question 6

Company	Yes	No
A	2	2
B	5	5
Other Respondents	5	3

Table 13 - Responses to Question 6

The responses to question four in the questionnaire brought about an interesting case as one respondent from company B believed that no one in the company practiced TDD however this I ruled as probably a mistake as during the interviews no one gave any impression of not believing they were practising TDD although two respondents did believe that there were some loopholes in the current TDD cycle.

Regarding the TDD cycle in company B it was clear from the responses that this cycle is determined by the QA team which was quite interesting because the QA's are primarily responsible for integration tests. Each development team (Backend or Frontend) has a team leader who is usually a senior developer that has been with the company for at least 3 years hence it is expected that he leads the team in regards to TDD principles or processes however this wasn't the case. Company A had no defined TDD principle and it was obvious from both questionnaire responses and interviews that although they did have some experience in TDD, they didn't adhere to any TDD principles at all.

5.5.2 Questionnaire answers and interview summary relating to RQ2

Below we will analyse two questions from the questionnaire which provide insights into research question two. The tables below show the questions and the response of the respondents.

Question No	Question	Company	No. of Respondents
8	**What do you consider the limitations of TDD in your company?**	A	4
		B	10
		Other Respondents	8

Table 14 - Summary of Question 8

COMPANY A

Limitation	Strongly Agree	Agree	Neither Agree or disagree	Disagree	Strongly Disagree
Increased Development time	3	1			
Insufficient adherence to TDD protocol	4				
Lack of developer skills in writing test cases	1	1	2	1	
Legacy Code			1	1	2

Table 15 - Responses to Question 8

COMPANY B

Limitation	Strongly Agree	Agree	Neither Agree or disagree	Disagree	Strongly Disagree
Increased Development time	3	5	1	1	
Insufficient adherence to TDD protocol	2	4	3	1	
Lack of developer skills in writing test cases	1	4	5		1
Legacy Code	3	2	2	2	1

Table 16 - Responses to Question 8

Other Respondents

Limitation	Strongly Agree	Agree	Neither Agree or disagree	Disagree	Strongly Disagree
Increased Development time	3	4	1		
Insufficient adherence to	1	2	3	1	1

TDD protocol					
Lack of developer skills in writing test cases	1	1	2	3	1
Legacy Code	4	2	2		

Table 16 - Responses to Question 8

The responses from Company A and B show that there is a strong conviction that the major limitation to TDD in their companies is the increased development time hence elimination of this limitation is something that is expected to bring about the full benefits of TDD. In company the failure to adhere to TDD protocol is also seen as a major block and this can also be deduced from the responses by Company B where 60% of the respondents believed this to be the case whilst only 10% disagreed with it. Regarding the developer skills in writing test cases, in company A and B there were no totally conclusive results as 50% of respondents neither agreed or disagreed with this assertion. The trend in the the reaction to Legacy code being a blocker was a bit different as in company A as 75% do not see this as a problem ideally because they most probably wrote the legacy code itself. In company B's responses to Legacy code as a blocker respondents differed in opinion. While 50% believed this to be a blocker, 30% of respondents believed this wasn't an issue and 20% were indifferent. Company B being an outsourcing company sometimes didn't build the projects from scratch but rather given the projects in a maintenance role hence the presence of legacy code in some projects. This was deduced from interview sessions where some interviewees confirmed having worked on already existing project maintenance while others maintained that they had never had only worked on projects from it's inception as opposed to maintaining an already existing project.

5.5.3 Questionnaire responses and interview related to RQ3

Below we look at the responses that are related to research question three in a tabulated form and then give a summary of the responses.

Question No	Question	Company	Respondents
7	What are the benefits of TDD to your company?(Select all that apply)	A	4
		B	10
		Other Respondents	8

Table 17 - Summary of Question 7

Company	Benefit	Respondents
B	Reduction in Defect Rate	5
	Increased Developer Productivity	0
	Higher Unit Test Coverage	6
	Reduced Integration Problems	3

Table 18 - Response to Question 7

Question No	Question	Company	No. of Respondents
9	**Will you readily recommend TDD to other companies?**	A	4
		B	10
		Other Respondents	8

Table 19 - Summary of Question 9

Company	Yes	Maybe	No
B	7	3	0

Table 20 - Responses to question 9

The results from the responses show that in Company B the biggest benefit of TDD is that it results in higher unit test coverage as 60% of respondents selected this option. Half of the respondents also believe that it results in defect rate deduction however, there seems to be some scepticism regarding if TDD actually leads to reduced integration problems as only 30% of respondents picked this option. The consensus however in the responses was that does not by any means increase developer productivity. When asked if they would readily recommend TDD to another company, No single respondent opted against recommending it. 70% percent of respondents said they would readily recommend. It is worth noting that the 30% that went for the maybe option were those who did not have TDD experience prior to joining the company and have worked in the company for less than 3 years

6 Limitations to TDD adoption

In this section I will give an overview of the limitations to the adoption of TDD in the industry. There are several factors which have limited the adoption of TDD in the industry however, in relation of this study I will look at seven categories according to the literature survey and responses from the questionnaire and the interviews. These limitations are as follows;

6.1 Increased development time

Development time in this context refers to the time taken for the implementation of the requirements (both functional and non-functional). Although it is relatively easy to measure the time taken in respect to software development, it however in the case of TDD largely depends on if the time used in corrective rework such as time taken to correct failure reports that arise from the later stages of testing, is actually captured into the development time[7]. For companies intending to adopt TDD, development time is always a huge consideration as in most cases it is considered to be a business-critical factor hence a loss in development time might over shadow the long term benefits of TDD adoption.

6.2 Insufficient TDD experience/knowledge

TDD experience/knowledge in this context refers to the level of experience of the developer(s) in respect to TDD. A lack of experience on the subject can hinder companies from adopting as this in most cases will prolong development time and affect developer productivity.

6.3 Lack of upfront design

Design here refers to the process of structuring the system that is to be built such that it doesn't result in architectural problems and by extension results in improved architectural quality. Currently there is no existence of massive empirical evidence that actually does support the fact that lack of upfront design is/can be a problem likewise there is also no massive empirical evidence that contradicts it also[13]. Since TDD focuses mainly on a small amount of design upfront which requires constant refactoring to meet the requirements this can result in increased development time also.

6.4 Domain and tool specific issues

This generally is related to the technical problems involved with the implementation of TDD as the methodology requires certain tool support (test automation frameworks) in other to be effective. Having the right tools is a quintessential factor in TDD, one that can positively or negatively impact its practice.

6.5 Lack of developer skill in writing test cases

Developer skills in this context refers to the ability of the developer to be able to write automated test cases which are efficient and effective[7]. Considering the fact that TDD as a development methodology emphasizes developers writing tests first before the implementation code, the implementation code will greatly depend on how good the test cases are hence the skills of the developer in regards to writing test cases is a huge factor the adoption of TDD relies on.

6.6 Insufficient adherence to TDD protocol

Adherence to TDD protocol simply means the extent to which the encompassing steps/guidelines on how TDD should be implemented. In experiments conducted[7] it was discovered that in some cases developers abandoned the TDD protocol due to several issues amongst which are time pressure, shortage of the perceived benefits of adhering to the guidelines and lack of discipline. It is worth noting that this observations were actually made at organizations where the preferred software development methodology IS TDD[7].

6.7 Legacy code

Legacy code in this study refers to codebase which is already existing and being passed down in the development organisation. TDD concept of TDD doesn't really encompass how Legacy code should be handled as it assumes that all code is developed from scratch. This can be quite problematic especially for big organisations which have a huge chunk of legacy code hence resulting in huge concerns about TDD adoption.

7. Answers to Research Questions

This research focused on an in depth analysis of test driven development as it affects the two major participating companies. During the course of this research we have analysed the different states and levels of TDD in both companies as well as the challenges faced in regards to it adoption.

A literature review was conducted in order to report the effectiveness, benefits and or drawbacks of TDD while an case study which focused on gathering data from participating companies included a survey and oral interviews to get information regarding TDD current practices and state in the both companies as well as voluntary participating companies.

This section will focus on mapping the results drawn to the relevant research questions. Each research question accompanied by a detailed answer can be found below.

7.1 State of TDD in both companies (A and B)?

***"RQ1**: What is the current state of TDD in both companies (A and B)?"*

7.1.1 Company A

Company A seldomly practices TDD. Occasionally they tend to boost their test coverage by focusing on key integration tests and important unit tests. Unlike Company B, there is no defined QA procedure hence the process is a bit more focused on fixing bugs if they are not found out during A/B testing and their unit test coverage is just a little below twenty-five percent (24.37%).

7.1.2 Company B

Company B have been practising TDD for approximately five years. They started with the practice a year after the company was established. At the moment they have a minimum threshold of a ninety-five percent (95%) unit test coverage for every project executed. They however do not have a specific figure for integration tests as the extent to which integration tests are done are determined by the QA assigned to that project hence this varies from project to project

7.2 Expectations of TDD in both companies?

***RQ2**: What are the expectations of TDD in both companies (A and B)?*

7.2.1 Company A

In company A there is no strong expectation regarding their current TDD practices. It was more of hope than expectation as the developers simply tried to follow TDD when they had the time to in the hope that this to a large extent will catch some bugs early. Their expectation regarding proper TDD implementation in the future was geared towards catching bugs early

as this is believed to eventually translate into defect reduction on their software. Currently there is almost always a quick rollback and bugfix after every deployment of a new feature to their staging and testing environment.

7.2.2 Company B

Company B has a broad vision of its expectations from TDD which although to a large extent they are currently satisfied with, they however, believe that there could be more benefits for them. In general they expect TDD to reduce the defect rate as it has actually occurred over the past five years (an average of 15% reduction in bug reports yearly). Ideally they would have expected to have a higher percentage in bug reports reduction, however, this expectation is managed as they are aware and admit that they do not practice TDD strictly like it ought to be even though there are already defined guidelines set. In certain situations the concept of TDD is abandoned if there isn't enough time to do this. To solve this problem the Team lead and QA's decided to capture testing time when estimating cards for the sprint about half a year ago and this they believe has tremendously helped in letting the developers implement TDD although this largely depends on the type of feature and how urgent the said feature needs to be rolled out.

7.3 How has TDD helped company B?

***RQ3**: How has TDD helped company B?*

7.3.1 Benefits for Company B

Despite obvious blockers and limitations, the general consensus in company B is that TDD has greatly improved their entire development process from writing code down to their release cycle. There still exists a huge room for improvement as was echoed by almost all respondents however, there is great satisfaction regarding the results yielded so far. Every interviewee who had worked at the company for 2 years or more spoke about the reduction in bug reports reducing every year and they put this down to yearly improvement of the TDD cycle. For the first quarter of 2017 they believe that factoring in testing time as part of cards (coding tasks) duration has greatly improved the quality of tests written as well as enabled them to reduce the limitations of increased development time. There was concern however, regarding how strict the process should be followed as some QA's believed that the process should be adhered to strictly while some developers held the view that focusing on these strict procedures result in a drawback for developers creativity as well as the integrity of the system design. Developers fear that current laid down style puts too much concentration on unit tests as opposed to system or integration tests.

7.4 Success Factors for TDD Introduction

***'RQ4**: What are the success factors for introducing TDD according to the literature?'*

7.4.1 Simple and Incremental Development

TDD uses a simple, incremental approach to software development[11] although it's simplicity can be argued depending on several factors[11] . TDD enables you to have a working system almost immediately and this can be considered one of the major success factors for it's introduction. Usually in the first iteration there isn't a lot of functionality however the functionality does improve as the development continues and this makes it less risky also compared to the risks involved when trying to build the entirety of the system at a go with the hope that it will work when all parts are put together.

7.4.2 Simpler Development Process

Software developers who use TDD are generally more focused[8] when compared to those who do not mostly because a developer using TDD's main concern is how to get the next tests to turn green[12]. They focus their attention on getting a small piece of the software to work as opposed to creating the software by doing a lot of upfront design. In the case of building a quite complex software that will involve several thousands of decision, it'll be simpler to make those decisions while developing the code instead of trying to make all the decisions correctly before starting to write code.

7.4.3 Constant Regression Testing

Regression testing is simple terms can be said to be self-defence against software bugs[12]. In Software development, a simple change to a microservice for example may have several unforeseen consequences throughout the entire project. This domino effect is quite popular in software development hence the importance of regression testing. Considering the principle of TDD where tests are run before code is written, in effect any change to the code that results in an undesirable effect will be instantly figured out when the full set of unit tests are run for that change[12]. This running of unit tests for every change in code will to a large extent prevent any regression surprises when the final product is handed over. With the constant regression testing the development team will have a working system at every iteration which enables the development team to easily respond to any changes in requirements[13].

7.4.4 Reduced Design Complexity

The approach to developing software using TDD greatly helps in reducing software complexity since the main goal in TDD is to only add the code to satisfy the unit tests[16]. In general software developers tend to be forward looking hence building flexible and scalable software that can easily adapt to the almost ever changing requirements and/or new feature that clients usually come up with. This flexibility comes at the price of complexity. With TDD the developers have a suite of unit tests and this allows them to quickly tell if a change in the code has resulted in some unforeseen circumstance hence boosting the developers confidence to make changes to the codebase. In the TDD process, developers will constantly

be refactoring code as this is a part of the methodology. Having the confidence to make major code changes any time during the development cycle will prevent developers from overbuilding the software and allow them to keep the design simple. Using TDD, it's hard to add extra code that isn't needed. Since the unit tests are derived from the requirements of the system, the end result is just enough code to have the software work as required[16].

7.4.5 Improved Communication

The ideas surrounding a software can hardly be explained or described with words or pictures. Words are often inadequate in explaining the complexities of a function of software component. Serving as a common language, unit tests are used to communicate the exact behavior of a software component with less ambiguities.

7.4.6 Improved Understanding of Required Software Behavior

Different projects, pose different levels of requirement, in most instances, these requirements are quite comprehensive and other times vague. The understanding of the desired behavior of a software, can be best acquired by writing unit tests before codes[16]. This is mainly because, the pass/fail criteria for the behavior of the software is added which builds the required knowledge of how the software must behave. Increasing the fidelity of required behaviors, adding more unit tests due to new features or bugs represents these required behaviors.

7.4.7 Simpler Class Relationships

A software can be said to be complete in designed only with well-defined levels and clearly defined interfaces between levels. This gives the possibility of an easier testing and the reverse also remains true[16]. Code writing through tests makes the focus very narrow, hence, reducing the use of complex class relationships. As a consequent, the code forms in little blocks and fits compactly together. A code which is difficult to test is generally a bad code. Similarly, if the code design is problematic then the unit test will be difficult to write. The main function of the unit tests is to help point out bad codes, problem modification for a better designed, more modular code[12].

7.5 How can the results be used to facilitate TDD introduction

1. ***"RQ5**: How can the results from the study be used to facilitate the introduction of TDD in company A?"*

There are several factors to be considered for before adopting TDD. From the literature survey certain key points have been identified. Understanding the dynamics of these key points and how they affect TDD is an important part of TDD adoption.

There is no consistent evidence in the literature reviewed that clearly says TDD supersedes TLD in terms of testing effort, the quality of code produced or the developer's productivity

most notably the experiments conducted in this context[8] however, there are strong cases where TDD can be seen to reduce defect rates in the development cycle[12]. TDD as a practice needs several consideration regarding the short and long term benefits for its adoption [11].

7.5.1 Developers current level

The benefits of TDD might not necessarily improve productivity to a high level depending on the experience level of the development team[20]. For young development teams the benefits can tend to be higher and the long term benefits greater than with more experienced development teams[22] hence this is an important factor to consider when introducing TDD to company A.

7.5.2 Impact of design

For software where the design is not totally clear from the start and evolves as the product goes along this will result in several changes to the test cases and that has the tendency to result in several breaking changes continuously[9]. Hence, for TDD to be adopted by Company A, special attention has to be paid to the impact it will have on their software design.

7.5.3 Huge Time Loss

When it comes to usual data structures and black box algorithms, unit tests would probably be perfect hence making this type suitable for TDD however algorithms which tend to changed or constantly tweaked/fine tuned, there is a huge time investment (or loss) and this might not necessarily be justifiable making this one of the more critical factors for consideration[7].

8 Proposed Adoption Solution

In order for Company A to adopt TDD to enable them get visible results during the adoption implementation phase, based on the results gathered from this study I am proposing the following:

8.1 Estimate testing time-factor into card/task estimation

Time happens to be one of the major concerns for company A and this factor was echoed in almost all the industrial studies reviewed and all the companies surveyed. Due to the fact that there are business goals to be met, we observed that time taken for actually implementing the TDD flow was never factored into cards estimation rather the implementation of the task was all that was accounted for.

As deduced from company A, TDD at inception might be a bit slow however once the developers are conversant with the approach and the time factor for going through the TDD flow is being considered when estimating cards/tasks then there is a higher possibility to get the cards done in time using the proper TDD flow.

Previous estimation process:

When planning sprints in company A, time for cards were calculated based on just two factors i.e. implementation and testing but this wasn't helping the company to engage in proper TDD. After a proper analysis of this estimation process in relation to the time taken we discovered that writing test cases took almost the same time as writing implementation code and also refactoring took at least half of the time needed for implementation hence the estimates were wrong almost all the time and this resulted in unfinished sprint which was blamed on the TDD process as being too tedious. Never factoring in the time taken to refactor the code was also a huge problem. This resulted in little or no refactoring in most cases as refactoring was done mostly as an afterthought. The drawbacks included a high ratio of bug reports for tasks completed, usually about 1:1 and low code quality (the readability of the code was quite low).

Adopted estimation process:

Based on the results from the study, considering the fact that company A like most startup companies do not have a QA team, estimation of the cards was done based on the knowledge of implementation details duration, meaning if the task would normally take 2-3 hours for implementation code alone then approximately 4-6 hours will be allocated in order to use the proper TDD flow as this accounted for both time involved in writing the test cases, writing the implementation code and also refactoring the code. This resulted in an improved weekly sprint planning and drastically reduced number of bugs. In less than a month as after the first two weeks sprints passed, it was discovered that just 4 code related bugs were found in production from 14 different cards implemented (although there were some graphical bugs).

This was a record low in relation to bug reports as it came to approximately 1:4 when comparing to the previous minimum ratio of 1:1 for bugs to sprint tasks.

Approaches we tested:

Before adopting the approach mentioned above at company A, we tried estimating the tasks based only on the knowledge that testing took as much time as writing the implementation code. This didn't yield a lot of positive results as the bug reporting was just still quite same thing with a slight improvement of ratio 1:1.4 (a 0.4 improvement). Also the sprint was only about 70% completed despite this. We wrongly assumed that this was due to the difference in the developer's different levels hence the following week we considered this approach for only the senior developer but the results had little or no significant change. This prompted us to revisit each step in the TDD cycle and estimate time taken for each step. During this review with the developers we discovered that refactoring was almost always never considered but still took approximately about half the time needed when implementing hence we factored this into the next sprint and afterwards saw a huge spike in improvements.

Conclusion:

In summary, planning the time-factor properly into sprint cards/tasks greatly helped company A in reducing the number of rollbacks after feature implementation hence saving them way more time with bugfixes when compared with the time estimated for a thorough TDD flow.

8.2 Clarify the concept of TDD

This study showed that there exists the misconception that TDD is seeing as an automated test booster as opposed to a development methodology. In company B we found out that some developers actually practice TLD as opposed to TDD because there is a heavy focus on "test coverage" which isn't the main goal of TDD. Although Natalia Juristo et al. [37] was not reviewed in the context of this study due to a low number of citations, based on their contributions to the field of TDD it is worth reporting that their results in the research they reported that the unit tests are almost never up to date. It will be indeed beneficial to Company A to understand that the concept of TDD isn't the percentage of test coverage but rather the process of implementing as little as is needed for the test cases to pass.

Previous concept of TDD in Company A:

In company A, the TDD process was seen to not really be beneficial as it was rarely used and even in occasions when it was claimed to have been used, the process wasn't totally correctly followed. There was a specific bias discovered in this regard because whenever the developers planned to do TDD they had always already designed the implementation code in their minds consciously before actually proceeding to write tests. This generally isn't what TDD preaches when it refers to Test first. Also another notion we discovered was that whenever developers decided to focus on writing the tests first following the proper TDD cycle they ended up just brute forcing the implementation code to make the tests pass. This

usually resulted in a lot of hacky solutions and the code quality in general wasn't good because it was difficult to read and assimilate.

Also there was the notion that the difference in developer skill set and experience level had a huge effect on how well they could implement TDD.

Concept to be adopted:

Although it is difficult to get rid of the previous notion of TDD, however clarifying the concept is vital. Getting rid of the previous notion requires the developers to be open minded and willing to change. At the moment this isn't something that can be measured but a way forward will be to specify the cards in a way that it is difficult to build a source code model already before writing tests because the cards are extremely explanatory hence by reading through you know what the card entails and this helps you to think in the tests first paradigm.

Also giving developers some sort of training on TDD occasionally has the tendency to improve the amount of tests they write although this improvement might not necessarily affect the quality of the tests.

Summary:

In company A we tried to get rid of the notion by explaining why TDD has to be done the right way and why it is important to actually think of the test cases first before actually thinking of the implementation code. By specifying sprint cards in great detail it became possible for developers to actually pick out use cases (as these were specified in the cards) to begin thinking of writing tests first and actually following it through hence removing the idea of actually first thinking about the implementation code in their head and designing the tests based on that.

Regarding developers skill and experience, all the developers at company A took a comprehensive TDD course related to the development stack in use but the impact wasn't significant as it didn't directly relate to any significant changes in bug reports or time taken to finish sprints tasks. The only change was that the volume of test cases which increased slightly by about 6% approximately.

8.3 Adhere Strictly to TDD

In order to reap the benefits of TDD, the TDD flow must be adhered to strictly. It is way better to take fewer cards into the sprint and have a fully functioning piece at the end rather than taking too many cards into the sprint and spending similar amount of time doing rollbacks and bugfixes as currently experienced in company A.

TDD no doubt is a tedious process and will involve a lot of patience from those implementing it but ultimately the more the developers become familiar with it the less tedious it will become. In order to see any effect from implementing TDD the methodology needs to be followed religiously in all cases regardless of the experience of the developers involved.

As is often the case, from this study we discovered that there were several lapses in TDD adherence even from companies who believe they practice strict TDD compliance. Strict TDD compliance will not only save time if factored properly but also improve the percentage of completed sprints as well as less time on bug fixes and rollbacks.

8.4 Adoption Summary

Based on all the the adoption suggestions given in this chapter we have come up with a roadmap for TDD adoption in the company. This roadmap involves the following key points:

1. It is imperative to give all developers some sort of training on TDD regardless of their experience level with it (most especially when onboarding new developers into the development team).
2. Clarify the essence of TDD with the development team. It is imperative that they all buy into the concept and understand what it totally entails in order to avoid misconceptions.
3. When specifying cards/sprint tasks endeavor to capture the different use cases in the cards description as this makes it easier for developers to think in a test first approach.
4. When planning sprint tasks it is important to factor in the time taken to write the tests, write the implementation code and also refactor. This ensures to a large extent that none of the steps of TDD are actually skipped due to lack of time.
5. Ensure that developers follow the TDD process strictly. Write down the guidelines of TDD, write down the importance of following this process and during sprint check-up endeavor to show the developers the improvements to code quality, test coverage et al as the case may be, no matter how little the improvement is. This serves as a source of motivation.
6. When extending or improving existing features, it is important to note that while writing the tests for the use cases, it is also imperative to change tests that should be affected by the change. This should not be done as a post coding activity.

9. Conclusion

For this thesis, a literature review and semi-structured interviews with industry professionals were conducted. The aim was to find answers to five research questions postulated regarding Test Driven Development implementation in early stage startup companies.

The first phase of the study focused on studying the software development process in Company A in order to get an understanding of what they do in regards to Test Driven Development i.e. if they use TDD, if yes, then how they actually do it. The second phase involved studying the TDD process implemented in Company to ascertain its benefits and/or drawbacks. The last phase of the study involved an in depth analysis of the information gathered from the first two phases as well as gathering information from volunteer companies and professionals that meet our selection criteria to see how it could be used as a possible guideline for TDD implementation in start ups by taking into consideration the hindering factors as well as the benefits.

For the literature survey, a total of thirty papers (30) were finally selected out from the initial search result to meet the criteria set out. Four (4) extra articles which are related to the case study but were neither conference or academic papers were taken into consideration also as they provided some information relevant to the study bringing it to a total of thirty four (34). Through the results gotten from the literature survey a questionnaire for the case study was designed.

After the conclusion of the literature survey and questionnaire design, interviews were conducted with volunteers from both companies. The interviews were then transcribed and analyzed using thematic analysis in order to gather meaningful results from the data. The interviews and questionnaire provided insights which were indeed useful for the study.

This research aims at creating an adoption strategy of TDD for startups (Company A in the context of this study) due to the strong desire for it but low success rate in adoption. Although it gives insights and proposes some good methods I however believe there is more room for improvement considering the fact that startup culture all over the world is growing rapidly and there is a strong need to find an easing solution through which startups can adopt TDD without a large amount of drawbacks.

We have strived to give some good strategies e.g. planning the proper time-factor into task estimation, not limiting TDD to only unit tests, strictly following the TDD cycle et al. which we believe can ease the adoption of TDD for startups who are willing to adopt this methodology. The study can be improved upon most significantly with a larger number of startup companies for the sample group.

10. Acknowledgment

First and foremost I will like to thank God for keeping me alive and enabling me to finish this thesis. To everyone who has supported me in one way or another throughout the journey of this thesis I will like a say I am indeed grateful. Special thanks go to my supervisors Dr. Dietmar Alfred Paul Kurt Pfahl and Kaarel Kotkas for the guidance and counselling during the entire thesis, I wouldn't have been able to do this without you. To Companies A and B for taking their time to grant me interviews and respond to the questionnaires despite their busy schedules and deadlines I say I am totally grateful and deeply honored. Special gratitude to the Software Development Team Lead of Company B for his support in answering all my questions and providing me whichever data or statistics I needed. To my aunty, Timipa Ebidou Gagariga, thanks for proofreading my thesis on several occasions and always giving me constructive criticisms and pushing me beyond my limits. Special gratitude to my mother, you have been my rock and my pillar and I will forever be grateful. To every member of my family that has supported my journey in the academia I am totally grateful.

11. References

[1] Thirumalesh Bhat , Nachiappan Nagappan, Evaluating the efficacy of test-driven development: industrial case studies, Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, September 21-22, 2006, Rio de Janeiro, Brazil

[2] E. Michael Maximilien , Laurie Williams, Assessing test-driven development at IBM, Proceedings of the 25th International Conference on Software Engineering, May 03-10, 2003, Portland, Oregon

[3] http://biblio.gdinwiddie.com/biblio/StudiesOfTestDrivenDevelopment

[4] H. Erdogmus, M. Morisio, M. Torchiano, On the effectiveness of the test-first approach to programming, IEEE Transactions on Software Engineering 31 (3) (2005) 226–237

[5] Lee Copeland (December 2001). "Extreme Programming". Computerworld. Retrieved January 11, 2011.

[6] Nagappan, N., Maximilien, E.M., Bhat, T. et al. Empir Software Eng (2008) 13: 289. doi:10.1007/s10664-008-9062-z

[7] A. Causevic, D. Sundmark, and S. Punnekkat, "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review," in Proceedings of the 4th International Conference on Software Testing, Verification and Validation (ICST), 2011.

[8] Davide Fucci, Giuseppe Scanniello, Simone Romano, Martin Shepperd, Boyce Sigweni, Fernando Uyaguari, Burak Turhan, Natalia Juristo, Markku Oivo “An External Replication on the Effects of Test-driven Development Using a Multi-site Blind Analysis Approach” in ESEM '16 Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, (2016)

[9] David Janzen, Hossein Saiedian, "Test-Driven Development: Concepts, Taxonomy, and Future Direction", Computer, vol. 38, no. 9, pp. 43-50, Sept., 2005.

[10] B. George, L. Williams, A structured experiment of test-driven development, Information and Software Technology 46 (5) (2004) 337–342, special Issue on Software Engineering, Applications, Practices and Tools from the ACM Symposium on Applied Computing 2003.

[11] Boby George , Laurie Williams, An initial investigation of test driven development in industry, Proceedings of the 2003 ACM symposium on Applied computing, March 09-12, 2003, Melbourne, Florida [doi>10.1145/952532.952753]

[12] Laurie Williams , E. Michael Maximilien , Mladen Vouk, Test-Driven Development as a Defect-Reduction Practice, Proceedings of the 14th International Symposium on Software Reliability Engineering, p.34, November 17-21, 2003

[13] P. H. Breivold, D. Sundmark, P. Wallin and S. Larsson, "What Does Research Say About Agile and Architecture?," in Fifth International Conference on Software Engineering Advances, 2010.

[14] http://derekbarber.ca/blog/2012/03/27/why-test-driven-development/

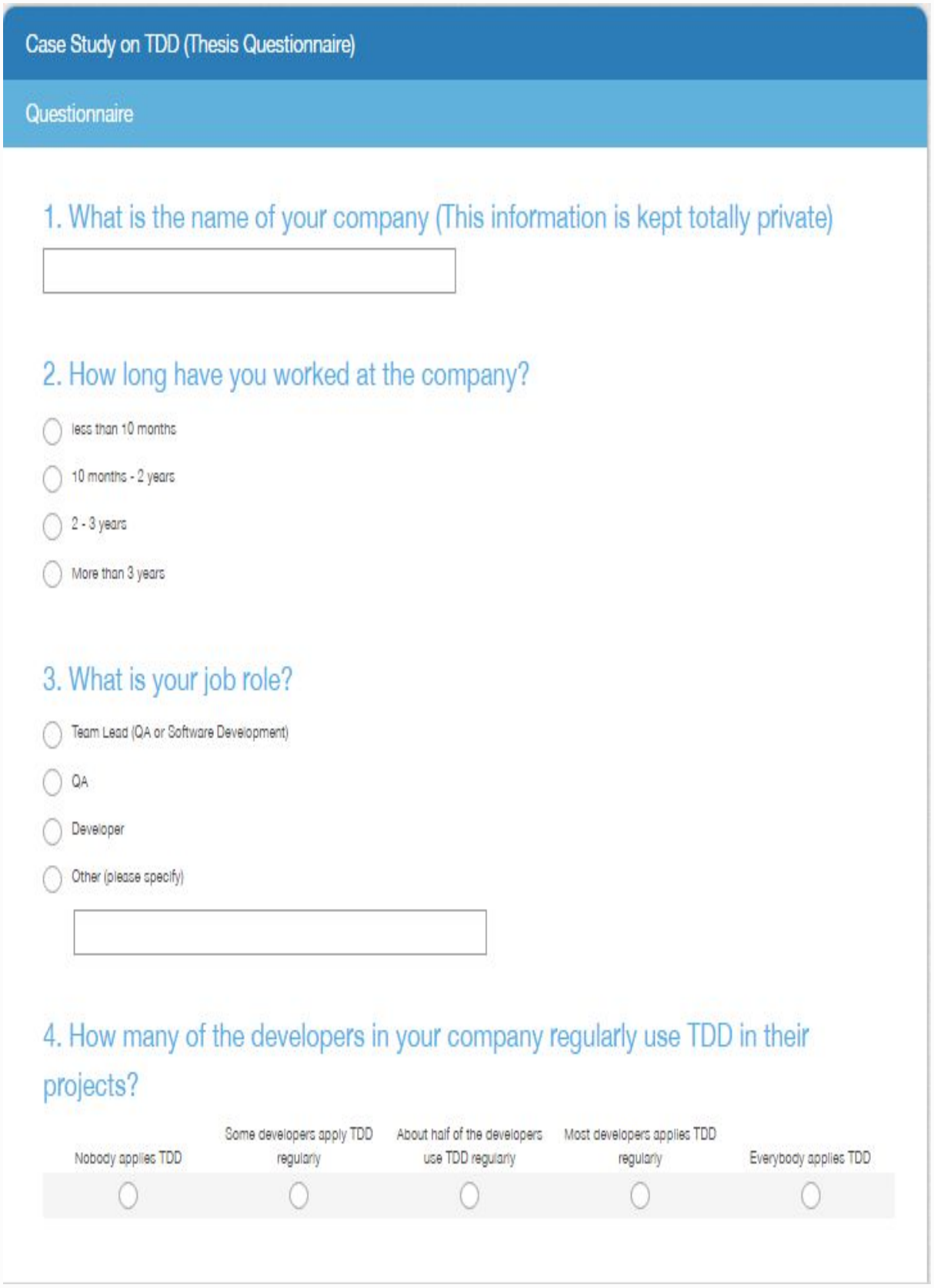
[15] http://wiki.c2.com/?TenYearsOfTestDrivenDevelopment

[16] Tosun, A., Dieste, O., Fucci, D. et al. Empir Software Eng (2016), An industry experiment on the effects of test-driven development on external quality and productivity doi:10.1007/s10664-016-9490-0
[17] Davide Fucci , Burak Turhan , Natalia Juristo , Oscar Dieste , Ayse Tosun-Misirli , Markku Oivo, Towards an operationalization of test-driven development skills, Information and Software Technology, v.68 n.C, p.82-97, December 2015 [doi>10.1016/j.infsof.2015.08.004]
[18] https://www.surveymonkey.com/r/HJ3HHNT
[19]http://www.zdnet.com/article/software-outsourcing-to-eastern-europe-which-countries-work-best/
[20] M. M. Müller and W. F. Tichy, "Case Study: Extreme Programming in a University Environment," presented at 23rd International Conference on Software Engineering, Toronto, 2001.
[21] A. Gupta, P. Jalote, "An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development", *First international Symposium on Empirical Software Engineering and Measurement*, 2007.
[22] M. Pancur, M. Ciglaric et al., "Towards Empirical Evaluation of Test-Driven Development in a University Environment", *EUROCON 2003*.
[23] Mäkinen S., Münch J. (2014) Effects of Test-Driven Development: A Comparative Analysis of Empirical Studies. In: Winkler D., Biffl S., Bergsmann J. (eds) Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering. SW QD 2014. Lecture Notes in Business Information Processing, vol 166. Springer, Cham
[24] Pedroso B., Jacobi R., Pimenta M. (2010) TDD Effects: Are We Measuring the Right Things?. In: Sillitti A., Martin A., Wang X., Whitworth E. (eds) Agile Processes in Software Engineering and Extreme Programming. XP 2010. Lecture Notes in Business Information Processing, vol 48. Springer, Berlin, Heidelberg
[25] http://david.heinemeierhansson.com/2014/tdd-is-dead-long-live-testing.html
[26] K. Beck, Test Driven Development: By Example, Addison Wesley, Reading, MA, 2003
[27] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - A systematic literature review," Information and Software Technology, vol. 51, no. 1, pp. 7–15, Jan. 2009.
[28] Oscar Dieste , Natalia Juristo , Mauro Danilo Martínez, Software industry experiments: a systematic literature review, Proceedings of the 1st International Workshop on Conducting Empirical Studies in Industry, May 20-20, 2013, San Francisco, California
[29] J. W. Creswell, Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, 3rd ed. Sage Publications, Inc, 2008.
[30] Runeson, P. & Höst, M: Guidelines for conducting and reporting case study research in software engineering, Empir Software Eng (2009) 14: 131. doi:10.1007/s10664-008-9102-8
[31] M. M. Muller and O. Hagner, "Experiment about test-first programming," IEE Proceedings-Software, vol. 149, no. 5, pp. 131–6, Oct. 2002.

[32] L. Huang and M. Holcombe, “Empirical investigation towards the effectiveness of Test First programming,” Information and Software Technology, vol. 51, no. 1, pp. 182–194, 2009.
[33] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C. A. Visaggio, “Evaluating advantages of test driven development: A controlled experiment with professionals,” in ISCE’06 - 5th ACM-IEEE International Symposium on Empirical Software Engineering, September 21, 2006 - September 22, 2006, Rio de Janeiro, Brazil, 2006,
[34] M. Pancur and M. Ciglaric, “Impact of test-driven development on productivity, code and tests: A controlled experiment,” Information and Software Technology, vol. 53, no. 6, pp. 557–573, 2011.
[35] P. Abrahamsson, M. Marchesi, and G. Succi, Eds., “Extreme Programming and Agile Processes in Software Engineering. 7th International Conference, XP 2006. Proceedings, 17-22 June 2006, Berlin, Germany, 2006, p. xii+228.
[36]L. Madeyski and Ł. Szała, “The impact of test-driven development on software development productivity—an empirical study,” Software Process Improvement, pp. 200–211, 2007.
[37] Romano, S., Fucci, D.D., Scanniello, G., Turhan, B. and Juristo, N., 2016. Results from an ethnographically-informed study in the context of test-driven development (No. e1864v1). PeerJ Preprints.

APPENDIX

I Questionnaire



Case Study on TDD (Thesis Questionnaire)

Questionnaire

1. What is the name of your company (This information is kept totally private)

2. How long have you worked at the company?

- less than 10 months
- 10 months - 2 years
- 2 - 3 years
- More than 3 years

3. What is your job role?

- Team Lead (QA or Software Development)
- QA
- Developer
- Other (please specify)

4. How many of the developers in your company regularly use TDD in their projects?

Nobody applies TDD	Some developers apply TDD regularly	About half of the developers use TDD regularly	Most developers applies TDD regularly	Everybody applies TDD
○	○	○	○	○

5. How long have you practiced TDD?

- ◯ upto 1 year
- ◯ 1 - 2 years
- ◯ more than 2 years

6. Did you have TDD experience prior to joining the company?

Yes	No
◯	◯

7. What are the benefits of TDD to your company?(Select all that apply)

- ☐ Reduction in defect rate
- ☐ Increased Developer Productivity
- ☐ Higher Unit test coverage
- ☐ Reduced Integration problems

Other (please specify)

8. What do you consider the limitations of TDD in your company?(Select all that apply)

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree	N/A (I don't practice TDD)
Increased development time	◯	◯	◯	◯	◯	◯
Insufficient adherence to TDD protocol	◯	◯	◯	◯	◯	◯
Lack of developer skills in writing test cases	◯	◯	◯	◯	◯	◯
Legacy code	◯	◯	◯	◯	◯	◯

Other (please specify)

9. Will you readily recommend TDD to other companies?

No	Maybe	Yes	N/A
○	○	○	○

10. Briefly describe your company TDD cycle

Done

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Kenigbolo Meya Stephen (date of birth: 27.05.1991),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

    1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

    1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Case Study of Test Driven Development,

supervised by Dietmar Alfred Paul Kurt Pfahl,
co-supervised by Karel Kotkas,
2. I am aware of the fact that the author retains these rights.
3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 18.05.2017