Bachelor Degree Project

Test-Driven Development with the Focus on Inexperienced Programmers: A Literature Review.

Authors: Adam Nyman & Oliver Rimmi
Supervisor: Dr. Rüdiger Lincke
Semester: VT 2022
Subject: Computer Science

Abstract

Test-driven development is a software development practice that prompts developers to write tests before writing source code. Studies report varied results on the effects that test-driven development has on the development process, and how this practice compares to other development practices, such as more traditional test-last development methodologies. There also seems like there has not been a discussion around the possible problems that a developer could encounter when adopting this technique, something that seems relevant to making accurate assumptions on the usability of the practice. A literature review was conducted, where the subject of test-driven development is examined with a focus on how inexperienced programmers interact with the practice and what effect it has on the product, in terms of external quality, productivity, number of test written and test coverage. The results suggest that there are no significant differences in external quality and productivity between TDD and TLD. The results also suggest that divide and conquer and refactoring are skills that ease the process of adopting the test-driven development practice.

Keywords: *Test-driven development, TDD, Test-last Development, TLD, Inexperienced developers*

Contents

1 Introduction

When developing software a great deal of developers probably, whether they know it or not, are using some form of so-called Test-last development practice (TLD). What is meant by TLD is that the developer first designs and implements code before later testing the software once it is considered *finished*. This process is rather straightforward and probably makes more sense, because how would one test something that is not done yet? That is where Test-driven development, also known as TDD, comes in. TDD is a practice where the tests are in focus and are therefore developed first. Thus, the rest of the development process is changed. A step of the development process such as the implementation instead is part of trying to make the earlier established test succeed.

During this thesis, TDD has been compared to TLD to observe the differences between the practices to hopefully be able to conclude whether TDD is a method that is worth learning, or if it does not lead to any improvements compared to TLD. The focus was further on novice developers and if or how TDD can assist them in improving their code. Additionally, the focus was on what possible difficulties novice developers can encounter when adopting the TDD practice.

1.1 Background

Test-last development is a term that seems mostly used within the context of discussions about test-driven development. The term is used to describe the more traditional way of programming that firstly implement code and features, and then writes tests in a later stage when a feature is considered finished. With this practice, the developers decide when it is appropriate to test implemented functionality. Thus, TLD could be an agile iterative development practice but it could also mean a more waterfall like development practice.

A contradiction to the Test-last development approach is the Test-driven development practice. This is a practice within software engineering in which developers write a test for a feature they want to implement before they implement the actual functionality. This is done by following three rules. The three rules are: i) Red - write a minimal unit test for a specific functionality or a part of the desired functionality. The test should fail since no code for that functionality has been added. ii) Green - write the minimal source code required to make the test succeed. iii) Refactor - improve the source code. These steps is repeated during the entire development process [1]. The idea is to follow the steps of this practice and thereby reduce possible fear in developers when taking on difficult and unfamiliar tasks, by allowing incremental steps towards completion and making developers more thoughtful about what they are going to implement [1, 2].

Development and IT-operations (DevOps) is currently a hot topic within the software development community and its practices have been adopted by some of the biggest companies in the world such as Google, Amazon, Nasa, Facebook, Netflix, and several more [3, 4, 5, 6, 7]. Given that these big and successful companies are using DevOps, other smaller companies will probably follow and that the trend of using DevOps will increase.

DevOps consists of multiple agile development practices which allows for short development cycles and a quick delivery of high quality software. The name *DevOps* represents the fact that there are two teams working together. One team is the *Dev* team and the other is the *Ops* team. The code developed by the developer team is sent to the operations team once the code has gone through some form of quality assurance. The operations team then deploys it and operates it [8].

How does DevOps relate to test-driven development? Currently it seems that TDD is one of the more popular ways of testing within the scope of DevOps [9, 10]. This means that if DevOps is on the rise, the use of the TDD practice will probably also see an increase. Hence, with the increase of TDD, the question of whether the practice is effective or not becomes generally relevant. Since, why should time and effort be invested into a practice that does not provide any valuable benefits?

To know if TDD is an effective development practice it has to be compared to other practices such as the TLD practice. This could be done in different ways. However, it seems rather common to compare effects such as the external quality and productivity, which are further explained in Chapter 2. Therefore, the main focus of this thesis was to examine these attributes when trying to uncover the effectiveness of the TDD practice. Additionally, it seems important to know if there are any difficulties that tend to occur when novice programmers adopt TDD and, if that is the case, try to discuss the most common of these difficulties. Since, the difficulties seem to be important because they may explain possible deviations in the results of different studies.

Earlier studies seem to demonstrate that the measured effects of these attributes vary. The majority of them also suggest that the differences in external quality and productivity are small and insignificant when comparing TDD to TLD [2, 11, 12, 13, 14, 15]. However, there have been more results published since these studies were conducted and it is therefore probably relevant to revisit these questions. The focus on possible problems that could occur does currently not seem to be discussed in other literature reviews and is thereby a topic that makes this thesis unique.

1.2 Related Work

The effects of TDD are a topic that has been under extensive research and continues to be to this day. All of the related works have conducted a systematic literature review regarding the effects that TDD has compared to more traditional development approaches [2, 11, 12, 13, 14, 15]. These articles mainly focus on the differences in external quality, productivity, and internal quality. The results of these literature reviews present that there are mixed results when it comes to external quality, productivity, and internal quality. Thus, there is currently not enough evidence to be able to claim that TDD is better or worse over these attributes compared to TLD [2, 11, 12, 13, 14, 15].

Kollanus, Munir et al., and Bissi et al. all conducted rather ordinary systematic literature reviews [11, 13, 12] and there is therefore not all that much to say about their methodologies. The main difference compared to this literature review is that they used Kitchenham as the evaluation methodology instead of PRISMA [16, 17]. Furthermore, they used additional databases for gathering resources.

Turhan et al. followed a methodology normally used for medicine when conducting the review and were therefore describing the process as if TDD has a *medical* effect on software development [15]. They tried to use this methodology to answer the generalised question: *"What does the published evidence say about the effects of using technique X?"*

[15]. The focus was further on quantitative data on effects such as external quality, productivity, and test qualities. The quantitative data was later summarised in tables where the different numerical values were presented to offer a clear overview of the results.

Y. Rafique and V. B. Mišić used the meta-analysis procedure for their review [14]. The idea of this approach is to summarise quantitative data and try to make a summarised result for their study. Therefore, additionally to the usual steps of systematic reviews they also had steps for standardising and un-standardising the data found in their study.

Shull et al. Their goal was to collect demonstrated evidence about the effectiveness of TDD compared to traditional development practices [2]. They chose to focus on delivered quality, internal code quality, and productivity, which are attributes that are similar but still different to what this thesis will focus on. They complemented their study data with commentary from Grigori Melnik, a senior program manager at Microsoft.

1.3 Problem Formulation

The knowledge gap for this project is based on multiple different aspects. Firstly, no literature review on how TDD compares to TLD that have been published later than 2016 was found. This means that any articles written after the date of that article have not been covered. Therefore, there could be information that has currently not been addressed by any of the other reviews and could possibly yield new and different results. Therefore, it seems reasonable to further investigate the subject including these new sources. Additionally, the literature reviews used as related works generalises the type of developer. That is different from what this thesis aims to do. This thesis focuses on the effects of TDD compared to TLD when used by inexperienced developers as well as what difficulties inexperienced developers can encounter when adopting TDD.

The reason that this specific gap was selected for this research was because of an article suggesting that developing software using TDD could help novices to build a better structure of the process and thereby achieve better results [2]. In this article, a senior program manager Grigori Melnik at Microsoft stated the following “He (Grigori Melnik) described working with a young, energetic programmer whose work unfortunately included many mistakes. Following TDD rigorously helped the programmer become more intentional in his work, thinking through the functionality he wanted to add. TDD does not just require skill and discipline; it also helps develop them” [2]. If this statement could be supported with more data and turns out to be true, it could encourage companies to hire inexperienced developers and let them work using TDD. This seems relevant due to many companies adoption of DevOps[3, 4, 5, 6, 7], which often utilise TDD [9, 10].

Another focus that seems to be unique, compared to the related works, for this thesis is the focus on what problems tend to occur when inexperienced developers start to use TDD. This focus has, from what has been observed, not been mentioned in the related works but is a point of view that could still be an important and relevant aspect of execution in a practice such as TDD. It would probably also be something that companies would like to know when considering to let inexperienced developers work using TDD.

With this gap and the fitting target group in mind, the following research questions were developed:

RQ1: What can be found in literature about external quality and productivity when inexperienced developers use TDD compared to TLD?

RQ2: What can be found in literature about test coverage and number of tests when inexperienced developers use TDD compared to TLD?

RQ3: What can be found in literature about the difficulties that an inexperienced developer can encounter when adhering to the Test-driven development practices?

These questions are the focus of this study and the questions that the result is based on and discussed around.

1.4 Motivation

The literature reviews used as related works covers papers in the subject of comparing TDD to TLD. However, they do not distinguish differences between what level of experience the participants have nor if this would have an impact on the results. Thus, it is unclear whether the results could differ for developers with different experience. Therefore, this thesis will focus on only one level of experience, namely, the student level of experience.

Another focus that the related works does not seem to highlight is that of difficulties that tend to occur when adopting TDD. However, this is a point of view that could be of importance. Maybe for any developer but especially for inexperienced developers that might tend to make mistakes more often. Additionally, the difficulties that could occur is also likely to be something that companies would like to know before they make a decision on what practice their inexperienced developers should follow.

This study aims to summarise other articles and their findings within the Test-driven development field of software engineering. The idea of summarising the articles is to hopefully be able to generalise the results to support a discussion about the field. Hopefully, this thesis will be a helpful guideline when it comes to novice programmers and decisions concerning Test-driven development and the adoption of this practice. It could be useful for a company to see how TDD can affect novice programmers as well as getting an understanding of what possible problems may arise when adopting TDD. Furthermore, it could also help teachers that look into TDD to know if it is worth teaching to students and how to aid them in their learning of this method.

1.5 Method

The methodology used when conducting this literature review was based on a paper written by Shaffril et al [18]. This includes seven steps that should be covered when conducting a systematic literature review. The steps are; *development and validation of review protocol*, *formulation of research questions*, *develop research strategy*, *quality appraisal*, *data extraction*, *data synthesis* and *data demonstration*. Additionally, snowballing was used as described in an article written by C. Wohlin [19] to find additional related articles. The methodology and its steps are further discussed in Chapter 3.

1.6 Results

This thesis presents a qualitative model regarding test-driven development and its effects on external quality in comparison to test-last development as well as other effects that can be observed in test-driven development compared to test-last development. Additionally, what difficulties inexperienced developers can encounter when adopting test-driven development is reported. This was done by conducting a literature review.

The data gathered during the data extraction process demonstrate how TDD affects external quality and productivity compared to TLD when used by inexperienced developers. Additionally, it demonstrates how TDD affects test-coverage as well as how many unit tests are written in comparison to TLD when used by inexperienced developers. Furthermore, the data presents what difficulties inexperienced developers can encounter when adopting the TDD practice. The results show that there are no significant differences when it comes to external quality and productivity. Additionally, they show that TDD improves test coverage and number tests written. Further, some of the difficulties identified are that TDD requires more thoughtful planning, students tend to spend insufficient time on the refactoring step of TDD and that it is difficult to use TDD when encountering complex and large problems.

The results were compared to previous similar literature reviews and studies. Additionally, the result was discussed in relation to similar previous literature reviews. The discussion and analysis of the results led to the possible conclusions as well as the identification of areas for future work.

1.7 Scope/Limitation

The scope of this thesis includes reviewing papers that have performed studies on how test-driven development compares to test-last development when used by inexperienced developers. The main component examined for this thesis is the observed results on external quality. However, there was also a focus on how other components are affected. Some of the other components examined could be productivity, test-coverage, and the number of tests written. Additionally, the focus was on what difficulties inexperienced developers can encounter when adopting test-driven development. The focus was also restricted to only inexperienced developers. Furthermore, the literature selected for this thesis was limited to articles written in 2012 or later.

1.8 Target Group

The target group of this thesis is primarily that of inexperienced programmers as well as companies that has a lot of employees that are inexperienced with TDD and maybe even inexperienced within the fields of programming and testing overall.

The idea of this thesis is to give the reader an idea of what will happen when adopting TDD, either to give a individual developer an idea of if it is worth it to adopt this technique and the hardships that comes with the practice, or to give companies and managers of development teams an idea of what to expect in terms of quality of their product and what problems that they could encounter if suddenly switching the development process from TLD to TDD.

Another group that is not the primary target group but could maybe find this thesis relevant is the academy and teachers that want to apply TDD in their teaching. This thesis could then give an idea of what to focus on in education to make the process of adopting TDD as successful as possible.

1.9 Outline

The rest of this thesis contains the following 7 chapters. In Chapter 2 the theoretical background is presented, this includes definitions and specifications of different terms that are important to understanding the thesis. In Chapter 3, the methodology is explained and discussed. The chapter will also contain reliability and validity discussions as well as ethical considerations. In chapter 4 the actual data extraction is explained in further detail. In Chapter 5 all the results are presented, this includes search results for the different search queries, the selected articles, and results for the different research questions. Chapter 6 is about analysis and specifying the validity of the arguments and statements found in the data extraction. Chapter 7 is the discussion. In this chapter the results are discussed. In the discussion, the result is also compared to the results of other studies. Conclusion and future work is the last chapter, Chapter 8. This chapter includes the conclusions that could be made from the results. It also contains suggestions for possible future works.

2 Theoretical Background

This chapter gives the theoretical background to the researched area and explains the different concepts and methods that are discussed in the thesis.

2.1 Test-Driven Development (TDD)

Test-Driven Development is a practice in software engineering where developers adhere to three rules. The rules are: i) Red - write a minimal unit test for a specific functionality or a part of the desired functionality. The test should fail since no code for that functionality has been added. ii) Green - write the minimal source code required to make the test succeed. iii) Refactor - improve the source code. This is repeated during the entire development process [1]. This development practice intends to split functionality into smaller pieces and test that every piece of functionality works as intended. Therefore the fear of breaking functionalities when refactoring or taking on difficult and new tasks is reduced for the developers. Additionally, it can make developers more intentional when writing code. Thus, TDD might be especially useful for inexperienced programmers which might run a higher risk of being fearful when developing and still have much to learn [1, 2]. Research on the actual effects of TDD compared to other practices is often inconclusive or ranges between a slight degradation to slight improvements when it comes to developer productivity, external quality and internal quality [2, 11, 12, 13, 14, 15].

The literature reviews on the subject often look at the effects of TDD in the context of developers with any level of experience. This differs from what this thesis focused on. The focus was instead on inexperienced programmers since they might run a higher risk of being fearful and could potentially benefit greatly from using a practice that forces them to split functionality into smaller pieces and test if it behaves correctly. Thus, reducing the chance of introducing bugs [1, 2].

2.2 Test-Last Development (TLD)

Test-last development is probably the more traditional way of developing code and making use of automated unit tests. As the *test-last* implies, testing is done after some functionality or all of the desired functionality has been implemented. The *when* in this case is likely decided by the developers. Thus, it could be an agile iterative development practice where some functionality is added and then tested but it can also mean a more waterfall approach, where all or a majority of the code is implemented first and then all of the code is tested at once in separate steps.

2.3 Development and IT-Operations (DevOps)

DevOps consists of multiple agile practices that allows for short development cycles and quick delivery of high quality software. The *Dev* in DevOps stands for development and the *Ops* stands for operations. Hence, the idea is that the developer team is working with developing new features and managing business requirements, while the operations team works with operating and managing the applications. The teams work closely with each other to get a clear understanding of the applications. The software developed by the developer team has to go through steps such as compilation and testing before it is delivered to the operations team. The delivery of code to the operations team is often done through an automated deployment pipeline [8]. One of the most popular ways to test the software during development is by using TDD [9, 10].

2.4 Unit Testing

A unit test consists of code written by a developer which focuses on checking whether a small part of functionality adheres to the requirements. This is used by developers to check if the source code behaves as intended. Since a unit test checks a small part of the functionality, many of them are required to be able to cover most or all of the functionality. When using multiple unit tests together it can be called a test *suite* [20].

2.5 Acceptance Testing

An acceptance test checks whether a system functionality behaves as the customer or in the case of a study, the researchers want it to. Thus, acceptance testing can be done by creating a unit test suite that adheres to the customers' or the researchers' demands or understanding of correct behaviour [21].

2.6 Test-coverage

Test-coverage is a measurement of how much of the code has been covered with tests [22]. There are different types of coverage, such as, statement coverage, branch coverage, and path coverage [23, 22]. However, the differences between the coverages are mostly just the scale. Given one, or several, of these different coverage types, a percentage of the tested statements, branches or paths can be calculated, giving a result on the coverage. When looking at, for example, the statement coverage to achieve complete coverage all of the statements have to be tested at least once. The same goes for the other types, but in those cases, it is for branches and paths respectively [23, 22]. Hence, test coverage is essentially a ratio between the number of tested items compared to the number of total items.

2.7 Regression Testing

Regression testing is a technique used to test the source code after a change of the source code has been introduced or if additional code has been added [24]. During regression testing, the existing test suites are run to detect if some functionality has been broken due to changes of source code or due to introduction of new code. There are several ways regression testing can be utilized. The most common way to utilize it is to use the *Retest All* definition [24]. This means that all the existing unit tests are executed. This can be very time-consuming. Thus, there are other definitions that reduce the number of unit tests run. The other definitions are; *Regression Test Selection*, *Test Case prioritisation* and the *Hybrid Approach* [24].

2.8 User Story

As described by Beck, user stories are scenarios written by the business to describe something that the system has to be able to do [25]. The idea with this concept is to share these simple stories in order to make sure that all developers know what is needed from the system as well as get the same idea of how it should operate [25]. In the context of this thesis, and the articles that data has been extracted from, the user stories are often not *real* user stories written for a client or business but rather artificial stories created by the researchers in order to explain the experimental tasks and the different features. Another reason why the user stories are important is since the features have to be clearly defined to later test the implementations for their quality and completeness.

2.9 External Quality

In the articles used for extraction of data for this thesis two different definitions of external quality is presented. The reason for there being two different definitions is because the versions shown in Equation 1 does not really consider productivity and external quality as separate attributes. Therefore some researchers have chosen to go with another standard in order to differentiate the two attributes from each other. This leads to the other way of measuring, shown in Equation 2.

According to the ISO 9126 standard, the definition of external quality is how well the written code adheres to the specified requirements or user stories [26]. Equation 1 demonstrates how this is calculated according to ISO 9126 standard [26, 27]. Where *Assert(Pass)* is the implemented functionality which behaves correctly and *Assert(All)* is all of the existing acceptance tests [26, 27].

$$QLTY = \frac{Assert(Pass)}{Assert(All)} \tag{1}$$

The second definition shown in Equation 2 is instead measured on every user story that has been tackled by a participant, rather than all user stories. A user story is considered tackled if a participant has implemented that user story with at least 50% correct functionality. The result of all tackled user stories is summed up and divided by the total number of tackled user stories. Equation 2 demonstrates how that is calculated [28, 29, 30, 31]. Where *#TUS* means tackled user story. The quality of every tackled user story is measured according to the previously mentioned ISO standard [26].

$$QLTY = \frac{\Sigma_{i=1}^{\#TUS} QLTY_i}{\#TUS} \tag{2}$$

The difference between these definitions is that the definition shown in Equation 1 does not consider whether the participants have tackled all the user stories. Except for that factor, the definitions are the same. Worth mentioning is that when adding productivity to the results, the differences between the definitions are even smaller and will therefore in this thesis be considered interchangeable.

2.10 Productivity

Productivity is a measure of how many functionalities participants have been able to implement during the time intervals of the studies that were used as sources for this thesis. The articles that were selected for this thesis does this in one of two different ways. The first way of calculating productivity is done by using Equation 3. In this equation the measurement of the productivity will result in a value between 0 - 100 and that all of the implemented functionality is compared in regard to the maximum number of acceptance tests [29, 28].

$$PROD = \frac{Assert(Pass)}{Assert(all)} \times 100 \tag{3}$$

The other way of calculating the productivity is using equation 4. This equation results in a value between 0 - 1.0. Additionally, a delivered user story is considered to have passed at least 50% of the created acceptance tests for that user story [30, 31].

$$PROD = \frac{Delivered\ user\ stories}{Total\ number\ of\ user\ stories} \tag{4}$$

The difference between the two ways is that Equation 4 only considers delivered user stories while Equation 3 considers all user stories, no matter their stage of completion.

2.11 Inexperienced Developers

Since *inexperienced developer* is a vague and subjective term it will here be explained what it means within the context of this thesis. In most cases, this means students or developers that still are learning and still have not mastered different skills in programming. However, in some instances in this thesis, it does not necessarily mean that the developers are still novices when it comes to programming, but rather just within the subject of test-driven development or testing overall.

2.12 Preferred Reporting Items for Systematic Reviews and Meta-Analyses Checklist (PRISMA)

PRISMA is a checklist that contains 25 topics that should be defined before and during the study [17]. These steps cover Administrative information such as thesis title and authors. An introduction with the rationale for the review as well as the objectives of the review. The methodology where the information sources, search strategy, study records, data items, outcomes and prioritisation, source bias assessment, data synthesis, meta-bias, and confidence in cumulative evidence is defined [17].

2.13 Critical Appraisal Skills Programme (CASP)

CASP is a quality appraisal evaluation tool in the form of a checklist [32]. The checklists contain multiple questions and criteria that could be used to evaluate the quality of the articles in question. Additionally, it can be used to identify possible bias in studies. At the point of writing this thesis, there exists checklists for *Randomised Controlled Trials*, *Systematic Reviews*, *Qualitative Studies*, *Cohort Studies*, *Diagnostic Studies*, *Case Control Studies*, *Economic Evaluations* and *Clinical Prediction Rules* [32].

2.14 Bowling Scorekeeper

Bowling Scorekeeper [33] is a task which a lot of the studies used in this thesis to let their participants implement using both TLD and TDD. The task requires the developer to implement an API for calculating the score of a player in a bowling game [29]. This task is considered to be a greenfield task, which means that the task does not depend on any specific system and is free for usage. Hence, it is possible to use this task in many programming languages.

2.15 Mars Rover

Mars Rover [34] is also a task which a lot of the studies used in this thesis to let their participants implement using TLD and TDD. The task requires the developer to implement an API which can move a vehicle on a two-dimensional grid [29]. This is also considered to be a green field task. Hence, it can be used in many different programming languages.

3 Method

In this chapter the methodology of the entire thesis is discussed. The chapter is divided into four different parts, namely: *Research Project*, *Research Methods*, *Reliability and Validity* and *Ethical Considerations*. In the Research Project, the overall structure of the project is presented. In Research Methods the different methods are presented in further detail. Reliability and Validity present the measures that have been made to make the research process replicable and to discuss the risks of interpretations or invalid influences on the result. Lastly, in the Ethical Consideration part the risks of bias and other risks or harm are further discussed.

3.1 Research Project

This thesis is a literature review with a focus on novice developers working with test-driven development (TDD). The goal was to try to find information about how the TDD methods affect the process of software development, and whether it is a method that developers should use more frequently or whether the method is not worth it and probably should be avoided in most cases.

To stay as objective as possible the methodology that was chosen is that of a systematic literature review. However, for formality reasons, this thesis will still be referred to as just a plain literature review. For the exact methodology the article “The abc of systematic literature review: The basic methodological guidance for beginners” was used as a reference [18]. In the article, the author writes about seven main steps that should be followed to make a qualitative literature review and it was these steps that were followed in this thesis.

Step 1: Development and validation of a review protocol (which can be found in the Appendix of this thesis). This protocol contains metadata of the thesis, this includes basic information such as title and authors, but also more detailed information like the different search queries for the different databases and how the data was validated in terms of quality. This step is rather iterative and should be updated multiple times during the development of the project. To validate the protocol and make sure that all the significant aspects were featured, the so-called PRISMA checklist was used. This checklist is further discussed later in this chapter.

Step 2: The second step was to formulate research questions for the thesis. For this step, the following questions were considered:

- What effect does TDD have on external quality for inexperienced programmers?
- How do the external quality effects of TDD compare to the external quality effects of other practices (e.g., BDD, ATDD, and Test-last)?
- How is the efficiency of code development affected by TDD?

However, these questions were mostly seen as primary, and later in the process, these questions seemed to be somewhat narrow and were therefore replaced with wider questions with a more reasonable scope. However, even though the questions were changed the topic and overall focus stayed the same. The updated questions and a discussion on those questions can be found in the Chapter 1.3.

Step 3: For the third step the systematic search strategy was developed. The first part of this step was to consider what databases to use. For this thesis, ACM and IEEE were chosen to be used as the primary databases as sources of information. These databases were chosen as they are both well-established organizations within the field of computer science. The database google scholar was also considered. In the case of Google scholar, it was not chosen because of its quality, but rather because of its possible width in its searches. However, no valuable articles were found using Google Scholar that had not already been found in any of the other two databases and therefore the decision was made that Google Scholar was unnecessary for this thesis and was therefore not used. The details about the search queries and other necessary information about the search is discussed in Chapter 3.2.

Another part of the search strategy is the inclusion/exclusion criteria. In the case of this thesis, much of this process was done using search queries. However, once the filtering by the search queries was done there were still quite a few irrelevant articles that were featured in the search results. Therefore, the abstracts of all the articles that were found in the search result were read and all articles that were considered irrelevant were removed. The articles that were removed because of irrelevance when either:

- *TDD* was used as a term for other contexts than *test-driven development*.
- The focus of the articles was on how test-driven development could be used for specific scenarios (rather than on how TDD affects software development).
- TDD was used as a validation tool for the teacher, rather than as a learning tool for the students (novice developers).

Step 4: The fourth step of the project was in the form of a quality appraisal. The idea of this step was to filter the selected articles in terms of quality. To measure the quality of the articles, the CASP checklists were the method that was mainly used [32]. Using these checklists all the articles were evaluated based on several different criteria. A majority of the articles scored rather well on these criteria and were therefore still considered for the next step. However, there were also a few articles that did not include a majority of the criteria and these articles were therefore excluded from the process. A discussion in more detail about the CASP checklists can be found later in Chapter 3.2.

When reviewing the articles the GRADE criteria were also considered and kept in mind. These five criteria are the following [35]:

- Risk of Bias in Methodology
- Risk of far-fetched Evidence
- Inconsistent Results in Studies
- Accuracy of Studies Results
- Risk of Publication Bias

None of the articles that had already passed the previous filtering seemed to have any major issues with these points. Therefore none of the articles were removed because of these criteria alone.

One last method that was used to test the articles for quality was to check the rankings of the journals in which they were published. This was done using the GII-GRIN-SCIE Conference rating database. This is a rating system sponsored by GII (Group of Italian Professors of Computer Engineering), GRIN (Group of Italian Professors of Computer Science), and SCIE (Spanish Computer Science Society) [36]. The objective of the system is to rank the different conferences and thereby gain an estimated rating of the quality of the articles published by that conference. A majority of the considered articles for this thesis were published by conferences that were categorised as Class 2 or Class 3 conferences and were therefore considered to be of good quality by the rating system. A discussion in further detail about the system and the results of the different articles can be found later in Chapter 3.2.

Step 5: Data extraction was the next step of the process. In this step, the data was collected from the selected and filtered articles. The idea of this step is to find all information that could be useful when trying to reason and discuss the research questions and objectives. Therefore, in this step, all of the filtered articles were read to obtain all necessary data. The focus when reading was on method, results, and discussions. However, in most cases, most of the article was required to be read to understand the whole picture. All of the data that the methodology and result chapters were summarised and collected in a document to gain a better overview of the data.

Step 6: The sixth step was data synthesis. In this step, the data collected in the previous step are used to produce a result, a discussion, and a conclusion.

Step 7: The last step that was performed was data demonstration. In this step, the main idea is to demonstrate how the project was performed and hopefully present it well enough that it could be replicated by another researcher if necessary. To make sure that is the case, the PRISMA checklist was once again used to validate the report and its content, to make sure that all necessary parts were represented. It is also worth mentioning that the result of this step is this report, the presentation of the project, its method, results, discussion, and other parts.

3.1.1 Snowballing

In addition to the search strategy presented in step 3, **snowballing** [19] was used to find additional articles. In snowballing the articles that were used for data extraction were also used to find additional related articles. In the article "Guidelines for snowballing in systematic literature studies and a replication in software engineering" the authors presents how to apply the snowballing process [19].

Step 1: in this process is to select a start set of articles and in this case, it was all of the articles that were used during the first data extraction.

Step 2: was to select all of the references of the start set which fulfilled the criteria of being published 2012 or later. Once this new set of articles were selected, they were filtered by checking for duplicates. Afterwards, the remaining articles were filtered by reading the abstracts. Once the articles were filtered by their abstracts, the remaining articles were filtered by their full content.

Step 3: was to select papers which were going to be included for data extraction. The selection was done by reading the full content of the remaining articles again. If new articles were found for data extraction, the new articles were used as the set for the next iteration of snowballing. If no new articles were found, the snowballing ended. Hence, snowballing could be done for many iterations. Step 4-7 of the systematic literature review process was also applied to the articles found through snowballing.

3.2 Research Methods

3.2.1 PRISMA Checklist

The PRISMA checklist [17] was chosen to validate the creation of the review protocol. The PRISMA checklist has 25 topics. By following these topics, the review protocol was developed. The review protocol can be found in the appendix.

3.2.2 Alternatives to PRISMA

An important discussion around PRISMA is why this specific methodology was selected for this thesis. Examples of other methods are ROSES and Kitchenham, which both are other evaluation methods of literature reviews [16, 37] and might have been other possible alternatives to use instead. Hence, why was PRISMA selected for this project over these alternatives?

ROSES was the first other evaluation standard that was considered besides PRISMA. Both the checklists of criteria for the ROSES evaluation and PRISMA evaluation were examined and a decision was made that the ROSES evaluation seemed to be somewhat superficial and was, according to us, not giving all that much guidance in the evaluation process. Therefore, ROSES was discarded as a possible option for this project.

When it comes to Kitchenham the choice is not as simple. What finally became the deciding factor was that PRISMA seemed more straightforward. The Kitchenham evaluation method seemed to be more up to interpretation. Thus, it was decided to go with PRISMA since it seemed like the room for wrongful interpretations was smaller.

3.2.3 Search queries and databases

The databases that were chosen for this project were IEEE Xplore, ACM Digital Library, and Google scholar. Google Scholar, was however cut from the process since none of the articles found by the query in Google Scholar were not already presented in the search result of any of the other two databases.

Once the databases were selected, the next step was to define keywords that could be used to construct the search queries. The main keywords that were first considered were TDD and inexperienced. To find as many sources as possible synonyms of inexperienced were defined. The keywords that were then considered were *novice*, *student*, and *intern*. The reasoning behind these keywords was that they were considered as groups of developers that probably could be assumed to have less experience and limited knowledge of software development. To makes sure that the articles used was not outdated, a time constraint was also used. This time constrain was that no article should be published before the year of 2012. When these keywords and time constraints were defined the actual construct of the search queries could begin.

The different keywords were considered and used in the construction of the search queries, the idea was to find a search result that included a reasonable amount of articles. Thus, all the articles could be considered and read if necessary. Therefore, the keywords were stated as necessary to be in the title. However, then the number of articles was insufficient. When using a query that searched for Test-driven development in the title and inexperience and its synonyms in the entire article the number of articles found was between 20-40 articles per query. The title and abstract were then read for all of these articles to check for their individual relevance.

The actual search queries that were used for the different databases can be found in the table 3.1. For the ACM Digital Library, two different queries were used; both of these are also featured in the table.

Database	Search Query
ACM Digital Library Search query 1	[[Title: tdd] OR [Title: "test-driven development"]] AND [All: novice inexperienced student intern] AND [Publication Date: (01/01/2012 TO *)]
ACM Digital Library Search query 2	[[Title: tdd] OR [Title: "test-driven development"]] AND [All: efficiency productivity] AND [All: novice inexperienced student intern] AND [Publication Date: (01/01/2012 TO *)]
IEEE xplore Search query	(("Document Title":TDD) OR ("Document Title":test-driven development)) AND (("All Metadata":novice) OR ("All Metadata":inexperienced) OR ("All Metadata":student))
Google scholar Search query	allintitle: "test-driven development" OR TDD novice OR inexperienced OR student OR intern

Table 3.1: This table includes all of the databases and which search query was used for each database. Note that there are two different search queries for the ACM Digital Library and this database therefore occurs in the table twice.

3.2.4 CASP

For this thesis CASP was used to evaluate the articles that were used for data extraction [32]. The first step in the evaluation using CASP is to find the checklist that is relevant for the article in question. There are different checklists for different types of studies, for example, systematic reviews, qualitative studies, and cohort studies [32]. Once the corresponding checklist has been selected the evaluation could begin. The first two questions are the most important ones, the idea behind these two questions is that if the answer is **no** for any of these questions the article in question can directly be discarded as it is not fitting as evidence for a thesis. For example, in the checklist for cohort studies, one of the checklists that were used for this thesis, the first two questions are “Did the study address a clearly focused issue?” and “Was the cohort recruited in an acceptable way?” [32]. When studying these questions it becomes rather obvious why the article could be discarded if these criteria are not met. The rest of the questions are then used to further examine the article and its quality. These questions are mostly on more specific parts and are more meant as a way to reason about the article and find any other possible issues. Once the article has been evaluated the task is up to the examiner to determine which articles that seem to be of higher quality and which articles should be considered but with a more skeptical approach, or maybe even discarded.

3.2.5 Alternatives to CASP

Just like with PRISMA it seems reasonable to discuss why CASP was used over other similar methods. The other methods that were considered were BMJ best practice's critical appraisal checklists and JBI's critical appraisal tools [38, 39]. Both of these methods were also checklists similar to those of CASP [32].

The reason that BMJ was neglected was that it did not have as many checklists for different types of articles as the other two, and more importantly, was missing important types such as Cohort studies and Quantitative which were the types that a majority of the relevant articles could be classified as. Therefore, these checklists were deemed unsuitable for this project and were discarded as a possible option.

When it comes to JBI the similarities between CASP are much smaller and both seemed like useful options. However, what finally made the decision was that CASP had an advantage in quickly filtering low-quality articles. CASP had a system where the first two questions had to be answered with a **yes**. If any of the two questions were not clearly answered by the article that article could quickly be discarded as it was deemed as unfitting as research data. Because of this quick filtering CASP was chosen as the checklist to be used for this project.

3.2.6 GII-GRIN-SCIE conference rating

As mentioned earlier in this chapter the GII-GRIN-SCIE Conference Rating, is a database where different conferences are ranked based on their quality and could therefore be used as a measurement of individual articles under the assumption that the article probably is of high quality if it is published by a high-quality conference. The conferences different categories are shown in table 3.2.

Class	Rating	Description
Class 1	A+, A++	Excellent, top-notch conferences
Class 2	A, A-	Very good events
Class 3	B, B-	Event of good quality

Table 3.2: Demonstrates the different grades for the GII-GRIN-SCIE system and what each rating means in terms of quality.

There are also conferences rated as C. However, there appears to be no description or classification of this rating. In the case of the articles that were found for this study, the following results were obtained: five articles from *Class 2* conferences, six articles from *Class 3* conferences, two article from a *C-rated* conference as well as 4 articles (all from the same conference) which were not found in the database. The journal that was not found in the database was called *IEEE Transactions on software engineering*. Since the articles are still published by IEEE we decided to keep them. Even though a rating could not be found in this database system. Additionally, the articles "Towards an operationalization of test-driven development skills: An industrial empirical study" [40] and "Findings from a multi-method study on test-driven development" [41] did not have a conference. Hence, no grade could be checked for those articles.

3.3 Reliability and Validity

Multiple measures have been made to increase reliability for this thesis. Firstly, all of the exact search queries and their results have been documented. This means that another researcher or reader could replicate the searches and verify that the search results are the same. The number of discarded articles as well as the titles of the kept articles are also documented. This is done for the same reason, that someone else could verify that they receive the same results.

When it comes to the collection of data the process was kept as systematic as possible. First, every individual article was summarised in a single paragraph, containing only the data that seemed relevant to answer the research questions and to obtain an overall view of the articles. For the next step, data were collected for RQ1 and RQ2. The data that was collected was then placed in a table that listed the articles and the different qualities for each of the questions. These questions were more objective than the last research question and could therefore more easily be presented in this form. The table of collected data Table 5.4 can be found in Chapter 5. Lastly, data for RQ3 were collected. This was done by reading the summaries to try and find points that could be found relevant to the question. These points and arguments were then collected in a list, which could also be found in Chapter 5. Since the collection was kept rather straightforward and systematic, the reliability is hopefully kept as high as possible.

Construct validity: Some of the theories examined in this thesis can have different meanings. Examples of the theories that could have different meanings depending on personal understanding could be *external quality*, *productivity* and *test-last development*. The mentioned theories along with other theories are described in the theoretical background chapter. Thus, the chance of misinterpretation is reduced. Additionally, an attempt to define and limit the number of general terms which are open for interpretation was made. This was done to improve the validity.

Internal validity: One of the primary concerns when it comes to internal validity and bias, is of course our own opinions as authors. Even though the data for articles have been chosen in a rather systematic and hopefully objective manner, there is still a selection and selection leads to bias. Something that for this project was considered unnecessary or irrelevant could be something someone else considers of importance. However, what is worth mentioning is that we, the authors, do not have any personal gain from being biassed either for or against Test-driven development, since none of us has any personal relationship to the practice. This is also a subject that was chosen by our own and therefore the results will not directly impact any specific company or university meaning that publication bias or any reason to search for specific data or results to please a specific audience is not present. Another important aspect of this type of bias is that there is in this case, not just a single author, this means that a discussion has been present in every step of the research and further reduces the possibility of biases.

External validity: Sources that were used for related works suggest that the results do differ when it comes to test-driven development studies in an experimental setting compared to an industrial setting [2, 11, 12, 13, 14, 15]. Hence, this could be a threat to the validity since this thesis is aimed towards the industry and most of the sources that are used for this thesis is in an academic setting.

3.4 Ethical considerations

A prominent ethical consideration when it comes to literature reviews are probably that of bias. Bias in literature reviews can come in many forms and this part will try to discuss as many of these as possible.

Internal bias and our, the authors, bias have already been discussed in the previous part Chapter 3.3. However, there are also other potential biases that are worth considering. One of these potential biases is that of other authors, the authors that have written the articles that have been reviewed. To keep this to a minimum, the CASP checklists were used to examine all the articles before using them as sources for this thesis. However, it is still not always possible to find these biases and there is a risk that this will affect the results in one way or another. For example, a majority of the articles used for data collection for this thesis are studies made on students by their professors. A majority of them state that they have in different ways tried to avoid bias, however, for example the knowledge of what the professors have told their students in advance or during their studies is limited and that could affect the results they gained. This is just one example of all the different biases that could occur but it also clearly presents how data could be affected in ways that are impossible to examine from the reports and literature alone.

An additional risk of bias in this thesis is that there are several authors of the articles which were used for data extraction that reoccurs multiple times. Some authors are present in more than half of the articles that were selected. If these authors have strong opinions about test-driven development, the results of those articles might be influenced by it. Thus, the risk of bias is increased in the thesis. However, the articles which ended up being used for data extraction had been run through a CASP checklist to identify possible biases. This means that something has been done in an attempt to reduce the risk of this type of bias.

Another ethical consideration that could be made about this thesis is the risk of harm. In this case, this risk is not that major. However, it is probably still worth mentioning. In this case, the possible risk would be that the study could alter the views of Test-driven development in either a positive or negative sense. Therefore, there is a chance of deterring people from adopting the TDD practice based on data that might be insufficient to be able to conclude that TDD is worse or better than TLD. However, the results demonstrate that completely deterring people from TDD or TLD is unlikely.

No other ethical considerations seem relevant to the thesis since the data was from already existing studies and this thesis just reviewed and extracted it. Thus, there is a low risk of sharing information that should be confidential. Additionally, no study data that are stored can be traced to a specific individual. Additionally, a study that involves external participants or test subjects was not conducted. Therefore, there was no need to consider if potential participants have given their consent for participation or not.

4 Research Project – Implementation

The initial step in extracting and collecting data for this thesis was to read all of the selected articles. While reading an article it was summarised in terms of the methodology, results, and conclusions and placed in a spreadsheet. The spreadsheet also contained other metadata such as the titles and references to every article. Thus, it was easy to keep track of what article included what piece of extracted data. The summaries also gave an overview of all the articles, making it easier to analyse and demonstrate the data.

Once all of the summaries were written, a table was formed containing some of the important aspects of the data, this was mainly to answer RQ1 and RQ2. The table presented the collected data for each of the two questions in different columns for every article that was examined. This was done to be able to easily present the data extracted for the research questions to the reader. The table was also used to keep the examination of the data more systematic and objective.

For the last research question RQ3, all of the summaries were read once again. This time with the purpose to find data only for this question. All the data that was extracted in this iteration was collected in a bullet-list where every piece of data was featured. Since the idea of this question was only to present possible issues with adoption of TDD it made sense to list all of the problems in a list. Presenting it in a way that quickly show all the findings.

Since this is purely a literature review there are no additional designs or implementations to further present or discuss.

5 Results

In this chapter all of the different results, obtained through the different methods, is presented. These different results include search results of the search queries, the results of the filtering by CASP checklists, and the results of the data extraction.

5.1 Results of the search queries

The first result that is presented is the result of the different search queries. The actual search queries and their construction is further discussed in Chapter 3.2.

The search results for the different queries were somewhat varied, however, to be able to filter the articles the queries were altered in different ways to make sure that the number of articles in each search was a reasonable amount. The exact number of articles found by each search query, as well as how many articles were used from each search can be found in table 5.1. Please note that the articles have to be unique to be considered. Several of the relevant articles were already found by the first search query. Therefore, the number of articles considered for the other queries was low since the articles were often duplicated in the searches.

Database	Articles found	Articles Considered Relevant
ACM Digital Library Search query 1	39	17
ACM Digital Library Search query 2	27	1 (most articles were already found by earlier query)
IEEExplore Search query	25	3 (most articles were already found by earlier queries)
Google scholar Search query	28	0 (none of the relevant articles were unique and were already found in the previous queries)

Table 5.1: This table includes all of the databases and the search results for each database. *Articles found* represent all articles found by that query while *articles considered relevant* represent how many of the found articles were used in the data extraction. Note that the reason for the great difference between the articles considered for each query is that every article was just considered once.

5.2 Results of the CASP

As shown in table 5.1, in total 21 articles were considered relevant after the initial filtering. Out of these 21 papers, 10 were removed after reviewing all sources using the CASP checklists. A majority of the papers that were removed in this step were because of the lack of a clear focus in the article. Either with vague and unclear research questions or in some cases a focus or research questions were missing altogether. Some of the articles were also removed because of missing results or methodology. None of the selected articles were removed because of being deemed to be biassed but rather because of the earlier mentioned factors.

5.3 Results of Snowballing

In Table 5.2 all of the articles which were selected through snowballing is shown. During the first iteration, 76 articles were found which fulfilled the criteria of being published in 2012 or later. Of those 76, only 5 were considered to be relevant to this thesis. During the second iteration 42 articles were found which fulfilled the criteria of being published in 2012 or later. Of those 42, many were duplicated from the articles found in iteration 1 and, none of the unique articles were considered to be relevant.

Title	Authors
Towards an operationalization of test-driven development skills: An industrial empirical study [40]	Davide Fucci, Burak Turhan, Natalia Juristo, Oscar Dieste, Ayse Tosun-Misirli, Markku Oivo
Test driven development: the state of the practice [42]	Susan Hammond, David Umphress
A dissection of test-driven development : Does it really matter to test-first or to test-last? [43]	Davide Fucci, Hakan Erdogmusand, Burak Turhanand, Markku Oivo, Natalia Juristo
Test case quality in test driven development: a study design and a pilot experiment [44]	Adnan Čaušević, Daniel Sundmark, Sasikumar Punnekkat
Findings from a multi-method study on test-driven development [41]	Simone Romano, Davide Fucci, Giuseppe Scanniello, Burak Turhan, Natalia Juristo

Table 5.2: This table includes all the titles and authors of the articles found during the snowballing process.

5.4 Articles selected for the data extraction

In Table 5.3 all of the selected articles are shown with titles and authors. These include only the articles that passed through both the initial filtering by search queries and the filtering by the CASP checklists.

5.4.1 Summaries of Selected Articles

Scanniello et al. conducted a study on 13 Master students with homogeneous knowledge in software engineering [45]. The study was performed by using a thematic analysis template in two sessions with a total summarized time of approximately 120 minutes. The results are based on the students' opinions of the subjects. The students were assumed to base their opinions on the experiences they had in a previous course where JUnit and TDD were used in comparison to the TLD practices they usually used [45].

Fucci et al. conducted a longitudinal cohort study on undergraduate students [28]. The study was conducted over a five-month period. The students were switching between TDD and their own programming style (the way they usually wrote code on their own) which according to the authors, are similar to iterative test-last development. Their research questions questions were; ***RQ1****: "To what extent do novice software developers retain TDD and how does this affect their performance?"* and ***RQ2****: "Are there differences*

Title	Authors
Students' and professionals' perceptions of test-driven development: a focus group study [45]	Davide Fucci, Burak Turhan, Natalia Juristo, Giuseppe Scanniello, Simone Romano
A longitudinal cohort study on the retainment of test-driven development [28]	Davide Fucci, Burak Turhan, Natalia Juristo, Giuseppe Scanniello, Simone Romano, Maria Teresa Baldassarre, Danilo Caivano
Results from an Ethnographically-informed Study in the Context of Test Driven Development [46]	Davide Fucci, Burak Turhan, Natalia Juristo, Giuseppe Scanniello, Simone Romano
An External Replication on the Effects of Test-driven Development Using a Multi-site Blind Analysis Approach [29]	Davide Fucci, Burak Turhan, Natalia Juristo, Giuseppe Scanniello, Simone Romano, Martin Shepperd, Boyce Sigweni, Fernando Uyaguari, Markku Oivo
Impacts of adaptive feedback on teaching test-driven development [47]	Kevin Buffardi, Stephen H. Edwards
Exploring influences on student adherence to test-driven development [48]	Kevin Buffardi, Stephen H. Edwards
Conformance factor in test-driven development: initial results from an enhanced replication [31]	Davide Fucci, Burak Turhan, Markku Oivo
Impact of process conformance on the effects of test-driven development [30]	Davide Fucci, Burak Turhan, Markku Oivo
Enhancing the Student Learning Experience by Adopting TDD and BDD in Course Projects [49]	Fabio Gomes Rocha, Layse Santos Souza, Thiciane Suely Silva, Guillermo Rodríguez
Investigating the Impact of Development Task on External Quality in Test-Driven Development: An Industry Experiment [27]	Natalia Juristo, Ayse Tosun, Oscar Dieste, Sira Vegas, Dietmar Pfahl, Kerli Rungi
A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development [50]	Burak Turhan, Natalia Juristo, Itir Karac

Table 5.3: This table includes all the titles and authors of the articles that were used in the data extraction.

between TDD and YW in the external quality of the implemented solutions, developers' productivity, and number of tests written?". They had two sessions where they let the participants train in ITLD and TDD. They let all participants work with four projects in both ITLD and TDD. The different projects used in the study were; *Bowling Scorekeeper*, *Mars Rover*, *Game of Life* and *SpreadSHeet* [28].

Romano et al. conducted an ethnographic study. The participants consisted of 14 students and 6 professional software developers from the same university [46]. The participants were divided into pairs. Each pair had a session with an observer for three hours. During these three hours, the pairs worked on a task called *MusicPhone* which all participants had some prior experience with. During each session the observer collected data such as notes, audio recordings of discussions and copies of various artifacts such as source code and notes. The goals of the study was to answer; ***how** practitioners and novice programmers perceive TDD*, ***how** they approach each phase of TDD*, ***why** they adhere (or do not adhere) to TDD*, ***why** they feel more comfortable with one or another of the TDD phases* and ***what are the characteristics** the developers believe an application must have so that they can successfully apply TDD* [46]. Thus, this article covers only our RQ3. Additionally, we discuss how the participation of professional software developers possibly affect the result in chapter 6.

Fucci et al. conducted an external replication of a previous study [29]. Their subjects consisted of 21 graduate students. They divided the subjects into two groups. One group used TLD on the *Mars Rover* task and TDD on the *Bowling Scorekeeper* task. The other group applied TLD on the *Bowling Scorekeeper* task and TDD on the *Mars Rover* task. Their research questions were; ***RQ1**: "Is there a difference between the number of tests written by TDD developers and TLD developers?"*, ***RQ2**: "Is there a difference between the external code quality of TDD developers and TLD developers?"* and ***RQ3**: "Is there a difference between the productivity of TDD developers and TLD developers?"* [29].

Buffardi et al. collected data from students submissions over two semesters at a university as well as conducted one survey on the students' perception of TDD and two questionnaires on the potential anxiety and fear connected to the assignments. The researchers focus were to observe how adaptive feedback would help students in the TDD process [47].

Buffardi et al. conducted a study on students over the duration of a semester at the university as well as a survey asking for the participants' perception of TDD as well as their own skills [48]. The researchers tested if test-driven development affected the test coverage of the assignment, as well as if TDD seemed to increase or decrease the amount of stress and anxiety that students had when working on an assignment [48].

Fucci et al. conducted an enhanced replication of a previous study [31]. Their subjects consisted of a mix of 41 undergraduate and graduate students. The subjects were trained in unit testing and TDD at six, three-hour sessions. The participants were randomly divided into two groups. One group applied TDD on a modified version of the *Bowling Scorekeeper* task. The other group Applied TLD on the same task. The focus in this study was if TDD results in more unit tests, if TDD results in higher external quality, if TDD users are more productive, if writing more tests increase external quality, if writing more tests increase productivity, if the conformance to TDD is linearly correlated to external quality and if the conformance to TDD is linearly correlated to productivity [31].

Fucci et al. conducted a study on their previously conducted controlled experiment [30]. The experiment had 58 subjects where 33 were graduate students and 25 were undergraduate students. The participants were randomly divided into two groups. The participants received 20 hours of training in unit testing and TDD. One group applied TDD on the *Bowling Scorekeeper* task and the other applied TLD on the same task. The research question of their study were; *What is the relationship between the conformance to TDD and its observed effects?* [30].

Rocha et al. conducted an experiment in form of a assignment in a university course [49]. The research questions that they tried to answer were ***RQ1**: "What do professors* *need to teach students to deal quickly and effectively with changing requirements during project development?"* and ***RQ2****: "How to evaluate individual performance when teaching a project based on TDD and BDD?"* The students were divided into five groups while working on the same assignment with each group consisted of 6-8 students. The different groups did decide by themselves which type of practice they wanted to use for the assignment. The options of practices that the participants got to chose from was Test-Driven development, Test-Last Development as well as Behavior-Driven development (BDD), which is a variant of TDD with a slight change of focus. Group 1 and 2 chose to be using BDD, group 3 was using BDD and TDD, group 4 was working with TDD, and group 5 was using test-last. The assignment was trying to replicate the "real world" by having a teacher acting as a client, and another acting as a coach for the teams. Group 5 had an initial good result in the earlier iterations. However, as time went on their implementation got more and more bugs leading to a decrease in delivery time. The researchers also claim that the students that used BDD and TDD had to plan the assignment more thoroughly and therefore could finish the project in time because they spent less time reworking the system [49].

Tosun et al. conducted an industial experiment with 17 participants with a focus on the differences between Iterative Test-Last Development (ITLD) and TDD [27]. All off these participants were industry professionals with different experiences in programming. However, they were still beginners in using test practices such as JUnit as well as in Test-Driven Development, this will be discussed further in Chapter 6. On the first day of the experiment, the participants were trained in ITLD and at the end were given a task to complete in three hours using ITLD. On the second day the subjects were trained in TDD and at the end were given a task to complete in three hours using TDD. Thus, all participants were slightly trained and then had to completed 2 different tasks in both ITLD and TDD. *The Mars Rover* task and *Bowling Score Keeper* task were implemented by the participants during this experiment [27].

Karac et al. conducted a controlled experiment on novice programmers in the form of university students [50]. The main focus of the study was further to compare the differences between fine-grained and course-grained requirements. All participants had to participate in lectures and exercises in how to program in Java, how to use eclipse and how to write JUnit tests. The actual study was a controlled experiment where the students were divided into two groups. In all the tests two different types of external quality were tested, namely "completeness" and "correctness".The correctness was based on how many of the researchers' tests were handled successfully. Whilst completeness was based on how many of the features were implemented. The tasks that were implemented by the participants were the *Bowling score keeper* task as well as the *Mars Rover* task. First one of the group were tested on a course-grained version of the BSK task and the other group was tested on a fine-grained version of the MR task. Then the group switched task and granularity. This way all participants executed both tasks and tried both granularity and both tasks were also tested for both types of granularity [50].

Romano et al. conducted an ethnographic study on 14 graduate students and six professional developers [41]. During the study, the participants worked in pairs. The participants were asked to implement a Java application which the participants had been introduced to prior to the study. Hence, all participants had a homogeneous understanding of the application. However, the participants had no previous introduction to the source code of the Java application they were asked to implement. The participants had also been introduced to TDD prior to the study. During the study, data was collected through notes, audio recordings of discussions, copies of code artifacts and through an automated tool in the participants IDE. The questions that they wanted to answer using the data were the following: *How practitioners and novice programmers perceive TDD*, *How they approach each phase of TDD*, *Why they adhere (or do not adhere) to TDD*, *Why they feel more comfortable with one or another of the TDD phases*, *How refactoring is carried out in TDD* and *Which characteristics the developers believe an application must have so that they can successfully apply TDD* [41].

Fucci et al. conducted a study that was performed on multiple different cites over the course of a 5-day workshop [40]. The study started off with a total of 43 participants which was reduced to 30 over the course of the study. During the beginning of the participants had workshops where they practised unit-testing, Test-driven development, pair-programming and refactoring. The actual experiment took place over a 4 hour session. The tests in the experiment were brown-fielded and were made to try and replicate problems from the real world [40].

Hammond et al. wrote a literature review about the current state of the test-driven development practice [42]. The idea of the review was to present the views and opinions of other authors in order to get a clearer view of the Test-driven development practice and its current state in the industry. The authors also looked at the TDD related practices BDD and ATDD. The review didn't do any kind of comparisons or summary of the collected data but was rather just stating and presenting the findings of the different authors to give a greater understanding of the subject [42].

Fucci et al. conducted a experiment that consisted of 4 workshops [43].The main objective was to look into the quality and productivity of software developed using the Test-driven development practice. In the research the researchers were also looking at how granularity, Uniformity, Sequencing, and refactoring effort impacted the quality and productivity when using the practice. The experiment used Mars Rover, and Bowling score keeper as well as a task called MusicPhone as the tasks that should be implemented by the subjects [43].

Čaušević et al. conducted a study that researched the differences between TDD and TLD in terms of test cases [44]. This was done using test-coverage, mutation score, and total number of failing assertions. In the experiment the subjects were asked to implement the bowling score keeper. The subjects were divided into two groups, one for test-first and one for test-last. All the subjects were master students at a university [44].

5.5 Data extraction

Authors	Effects on external quality and productivity	Effects on test-coverage and number of tests
Scaniello et al [45]	* External quality improved when using TDD. * Bugs are found early when using TDD. Thus, improving short-term productivity.	
Fucci et al [28]	- No significant difference in external quality. - No significant difference in productivity.	- Amounts of written test cases increase when using TDD.
Fucci et al [29]	- No significant difference in external quality. - No significant difference in productivity	
Buffardi et al [48]		- Improved test coverage when using TDD.
Fucci et al [31]	- Minimal reduction of external quality when using TDD. - Slight reduction in productivity when using TDD.	- Significant increase in the number of tests written.
Fucci et al [30]	- No significant difference in external quality. - No significant difference in productivity.	- Tends to write more tests when using TDD. - Greater test coverage when using TDD.
Tosun et al [27]	- Significantly lower external quality when using TDD.	
Papis et al [51]	- Significant increase in external quality when using TDD.	- Greater test coverage when using TDD.
Čaušević et al [44]		- No significant difference in test coverage.

Table 5.4: This table contains data from the data extraction for RQ1 and RQ2. The data is represented using two different bullet points. Firstly, there is the hyphen symbol (-). This symbol is used for measured effects or data. This is data that are generally more objective and are directly measured. The Asterisk symbols (*) are used for perceived effects. This includes answers from surveys, or opinions from participants or researchers. An empty cell in the table means that this was not discussed by that specific article.

In Table 5.4 the extracted data is shown for every article in regards to the first to research questions, RQ1: *What can be found in literature about external quality and productivity when inexperienced developers use TDD compared to TLD?* and RQ2: *What can be found in literature about test coverage and number of tests when inexperienced developers use TDD compared to TLD?* . Thus, the extracted data featured in Table 5.4 contains both objective results of measured comparisons when it comes to effects like external quality, productivity, amount of tests created, test-coverage, as well as subjective results regarding the studies participants' opinions and perceptions on TDD vs TLD.

To try and answer the third research question RQ3: *What can be found in literature about the difficulties that an inexperienced developer can encounter when adhering to the Test-driven development practices?* the following data were extracted from the selected articles:

- Students do not perform refactoring as often as TDD requires and consider refactoring to be less important than other phases [46, 45, 41].

- Promotes sloppy code, i.e., The students just want to get it working rather than spending time on ensuring quality and refactoring [45].
- Students tend to see Unit tests as a confirmation that they have achieved the wanted result, rather than as a part of the process. This can lead to test cases not being properly updated and kept in an unfinished state [46, 41].
- Inexperienced TDD users tend to "first build in their mind a sort of model of the source code to be implemented and only then write test cases" [41]. In these cases, rather than being a helpful tool with its normally concrete and systematic steps, TDD could become an obstacle because of how the developer instead needs to figure out tests to validate the implementation that they already have in mind [46, 41, 42].
- TDD requires a more thoughtful planning stage at the beginning of the process compared to TLD [49].
- A TDD user has to possess the skill of divide and conquer to use TDD effectively when tackling larger and more complex problems [50, 40].
- Granularity is important for the process of TDD which means that larger problems might be hard to solve using TDD [50, 43].
- It is difficult to follow the traditional steps in TDD and if the steps are not thoroughly followed, the benefits of TDD could be lost [42].
- It could be hard for the developer to know if the task is done, or whether there are still functionality missing [42].

6 Analysis

In this chapter the results from Chapter 5 is analysed to try and answer the research questions mentioned in Chapter 1.3, namely:

RQ1: *What can be found in literature about external quality and productivity when inexperienced developers use TDD compared to TLD?* For the analysis of the results on external quality, it was deemed that productivity would be considered as a part of external quality. The reasoning behind this decision is that in some articles the definition of external quality and productivity were the same as the ISO 9126 [26] standard for calculating external quality. Therefore, external quality and productivity are considered as the same attribute, *external quality*, to make the results of the articles more comparable.

There are a few topics found in three different articles that are worth examining in more depth since these topics could possibly impact the validity of the results.

The study made by Tosun et al. features only professional developers and this is something that needs further discussion since it could be seen as outside the scope of this thesis which solely focus on inexperienced developers [27]. However, it was still kept and deemed relevant because of the fact that it is stated in the article that the participants did not have knowledge about testing, such as JUnit and test-driven development and other practices. Therefore, even though not all of them were novices when it comes to programming they were still novices when it comes to testing and might therefore still have a hard time adopting TDD.

The study of Scaniello et al. is the other article that differentiates from the rest [45]. The results of this article are based on opinions of the participants rather than measured data. Therefore, the risks of potential biases does probably greatly increase and the results should therefore probably be examined from a slightly more sceptical perspective.

One last aspect that is worth mentioning is that papis et al. [51] does not specify that the ISO standard has been used to calculate the external quality. However, from the description of how it was calculated it seems like the definition is the same, just not explicitly stated by name.

Most of the articles claim that the difference in external quality as well as productivity is insignificant or minimal when comparing TDD to TLD. The only measured exceptions are the study conducted by Tosun et al. [27] which observed significantly lower external quality and Papis et al. [51]. which observed a significant increase in external quality. Additionally, Scaniello et al. [45] claim that the external quality is *perceived* to be improved when adhering to TDD. Hence, since a majority of the articles which objectively measure external quality states that there is a minimal or insignificant difference while the other two articles which observed significant differences contradict each other, it is probably safe to draw the conclusion that there is no significant difference in external quality.

RQ2: *What can be found in literature about test coverage and number of tests when inexperienced developers use TDD compared to TLD?* The effect on the number of tests written is mentioned in a three articles [28, 31, 30]. In all of these cases, there is an observed increase in the number of tests written when using TDD compared to TLD and none of the found sources state the contrary. Additionally, one of these articles observed a significant difference. Thus, it can likely be stated that TDD leads to a higher number of tests compared to TLD when utilized by inexperienced developers.

Additionally, test coverage is mentioned in four papers [48, 30, 51, 44]. Three of these papers observed an increase in test coverage when using TDD compared to TLD and one observed no significant difference. Therefore, based on the found sources, it is probably safe to say that TDD can lead to higher test coverage.

All the data presented in the result chapter regarding RQ3: *What can be found in literature about the difficulties that an inexperienced developer can encounter when adhering to the Test-driven development practices?* was more subjective compared to the other research questions where the majority of the data was measured objectively. This means that these arguments do not have as much credibility as some of the results that are backed up by objective measurements.

A topic worth mentioning for this research question is that the article written by Romano et al. [46] conducted a study where some of the subjects were professional developers, rather than students. However, since this article only covers research question 3 and that question is not directly based on measurements, but rather has the purpose to present possible difficulties that can be encountered when adopting TDD. Hence, it was deemed that the professionals likely do not impact the result since the difficulties encountered by professionals are probably even more likely to be encountered by inexperienced programmers as well. Thus, article [46] was kept even though a few professionals were present.

Another topic of analysis for RQ3 is that of Behaviour-driven developments (BDD) impact on the result in the study conducted by Rocha et al.[49]. Since this thesis only addresses test-driven development and test-last development, it was deemed that BDD was outside of the scope of this thesis. Therefore when looking at the result of this article it was only group 4 and group 5 in the experiment that were considered. Since group 4 used TDD and group 5 used TLD. The result and discussion concerning groups 1-3 which adhered to behaviour-driven development, or a mix of multiple practices, were not considered for this thesis.

This type of research question does not have a specific conclusion, but rather to highlight potential problems. However, the results could still be analysed by presenting common themes among the problems and suggest possible solutions. The common themes are that refactoring and unit testing is not done properly, that large problems are hard to solve using TDD and that developers have a hard time adhering to the process of TDD. Given these difficulties, inexperienced programmers could probably practise skills such as divide and conquer, refactoring and unit testing in order to achieve better results using TDD.

7 Discussion

In this chapter the results from Chapter 5 are discussed for each of the research questions.

RQ1: *What can be found in literature about external quality and productivity when inexperienced developers use TDD compared to TLD?* The results suggest that the majority of studies present no significant differences in external quality when using TDD vs TLD [28, 29, 30]. However, only one article observed a significant reduction in external quality and one article that claims a perceived increased effect in external quality [27, 45]. This means that the results are varied when it comes to external quality. However, it is worth noting that in the case where a significant effect was measured, the researchers do not differentiate external quality and productivity since their measurement of external quality does not take into account whether all user stories have been tackled or not. Therefore their results may be less credible which further suggests that the difference between the two practices in terms of external quality is insignificant. Which is in line with the results of previous literature reviews made within the field [2, 11, 12, 13, 14, 15].

In the study where a significant reduction was observed, the authors mention that their result is a contradiction compared to their previous study [27]. The contradiction seems to be due to the fact that the granularity of the tasks is important for the TDD practice to be effective. This assumption correlates to other articles that also discuss how granularity might be of great importance when it comes to TDD [50, 43]. Because of granularity and its impact on the external quality, it is probably safe to say that the result in the more recent study by Tosun et al [27], could also be valid even though it differs from the majority of the other results because of granularity being a more prevalent factor in that study. Granularity and its impact on TDD is further discussed later in this chapter.

RQ2: *What can be found in literature about test coverage and number of tests when inexperienced developers use TDD compared to TLD?* The number of tests written are mentioned in multiple articles. In the case of TDD, the number of tests significantly increased compared to TLD [28, 31, 30]. Additionally, the test coverage was improved when using TDD instead of TLD, according to the majority of the found articles [48, 30, 51]. Only one article stated that there were no difference in test coverage [44]. No related works were found that discussed the number of tests or test coverage in particular. Therefore, it is hard to say if this is a reasonable result. However, because of TDD's test-heavy approach, it makes sense that the focus is also on the number of tests written. It also makes sense that when the number of tests increases the test coverage is also extended, given that the tests are meaningful and not just a high quantity of similar tests. Thus, TDD would probably be more useful if regression testing is something that is planned to be utilized.

RQ3: *What can be found in literature about the difficulties that an inexperienced developer can encounter when adhering to the Test-driven development practices?* Even though some sources suggest that TDD is a useful development practice for inexperienced developers because of the required extra planning, that does not mean that TDD is problem-free for a novice programmer. As shown in the result chapter, students tend to have a hard time with large problems which do not have a fine-grained description. Thus, the main difficulty with the large tasks seems to be the ability to split them into smaller parts [49, 50]. Hence, a developer who has great divide and conquer skills would probably have an easier time using TDD. This together with the previously mentioned effects of task granularity on external quality demonstrates that for TDD to be successful, large tasks have to be well explained or well divided by developers.

Another problem that students tend to have when adopting TDD is to prematurely consider the work as completed. TDD's three steps of implementation tend to assure the developer that they are done before the task has been completed with high quality. An example of this is that multiple sources suggest that students do not perform enough refactoring of the implementation. Instead, they consider the work done at a premature stage [46, 45, 41]. This in turn leads to *sloppy* code since the focus is just on finishing the task and not to ensure quality in the product [45]. Another similar problem is that students tend to think of Unit tests as a confirmation that the task is done once the tests are no longer failing. However, this is also a dangerous assumption as this could lead to testing cases not being properly updated and left in an unfinished state, leading to a loss in quality [46, 41]. It is therefore important to note that for TDD to be successful, refactoring and updating of test cases must be done properly otherwise valuable qualities of the product could be lost. Therefore, for an inexperienced developer, writing test cases and refactoring are qualities that might be worth focusing on and attempting to master to become a great TDD developer.

The last problem to discuss is the fact that inexperienced TDD users tend to "first build in their mind a sort of model of the source code to be implemented and only then write test cases" [41]. In these cases, rather than being a helpful tool with its normally concrete and systematic steps, TDD could become an obstacle because of how the developer instead needs to figure out tests to validate the implementation that they already have in mind [46, 41, 42]. This problem slightly differs from the other problems, since this might not be an issue only for inexperienced developers. Unfortunately, no sources were found to support this reasoning, but it does not seem unreasonable to suggest that the more a developer has worked with traditional TLD the bigger the risk of still keeping the traditional mindset of implementing first and validating with tests later.

8 Conclusions and Future Work

The results, just as the related works, seem to suggest that the differences are insignificant when comparing TDD and TLD over the attributes of external quality and productivity. External quality also seems to be highly related to task granularity. Therefore, studies regarding comparisons over external quality and productivity should probably be avoided in the future and use resources for other questions instead. Some suggestions on what to research instead are probably how granularity affects TDD and in which tasks TDD is effective. Thus, the possible knowledge contribution becomes more about when TDD could be more effective compared to TLD rather than if. Another possible research that could be of value is to further examine the differences between professional TDD developers and other professionals in the development field to see if the differences are more significant in those scenarios.

Additionally, the number of tests and test coverage seem to be improved when using TDD instead of TLD. Just a few of the sources selected for this thesis mentions test cases and test coverage and as mentioned, the results of all but one of these articles claim to have seen a positive increase in both the number of tests written as well as in test coverage. Therefore, in the future, it may be more reasonable to study the subject of testing and test coverage in a TDD setting instead of doing more testing on external quality or productivity.

When adopting TDD *refactoring*, *dividing and conquering*, *updating unit tests* and *truly testing before implementing* are some of the reoccurring difficulties for inexperienced developers. Thus, when mastering these skills and mindsets a developer could properly learn to adopt TDD in a more successful manner.

References

[1] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[2] F. Shull, G. Melnik, B. Turhan, L. Layman, M. Diep, and H. Erdogmus, "What do we know about test-driven development?" *IEEE software*, vol. 27, no. 6, pp. 16–19, 2010.

[3] Y. Demchenko, Z. Zhao, J. Surbiryala, S. Koulouzis, Z. Shi, X. Liao, and J. Gordiyenko, "Teaching devops and cloud based software engineering in university curricula," in *2019 15th International Conference on eScience (eScience)*, 2019, pp. 548–552.

[4] "6 companies that are doing devops well," Jan 2018. [Online]. Available: https://www.helpsystems.com/blog/6-companies-are-doing-devops-well

[5] A. Duran, "Which companies are using devops?: Prolifics." [Online]. Available: https://www.prolifics.co.uk/which-companies-are-using-devops/

[6] C. Null, "10 companies killing it at devops," Jul 2021. [Online]. Available: https://techbeacon.com/app-dev-testing/10-companies-killing-it-devops

[7] "Top 5 companies using devops in 2020: Devops applications," Dec 2021. [Online]. Available: https://www.edureka.co/blog/companies-using-devops/

[8] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of devops concepts and challenges," *ACM Comput. Surv.*, vol. 52, no. 6, nov 2019. [Online]. Available: https://doi-org.proxy.lnu.se/10.1145/3359981

[9] R. T. Yarlagadda, "How devops enhances the software dévelopment quality," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 7, pp. 2320–2882, 2019.

[10] S. Rafi, M. A. Akbar, S. Mahmood, A. Alsanad, and A. Alothaim, "Selection of devops best test practices: A hybrid approach using ism and fuzzy topsis analysis," *Journal of Software: Evolution and Process*, vol. 34, no. 5, p. e2448, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2448

[11] S. Kollanus, "Test-driven development-still a promising approach?" in *2010 Seventh International Conference on the Quality of Information and Communications Technology*. IEEE, 2010, pp. 403–408.

[12] W. Bissi, A. G. S. S. Neto, and M. C. F. P. Emer, "The effects of test driven development on internal quality, external quality and productivity: A systematic review," *Information and Software Technology*, vol. 74, pp. 45–54, 2016.

[13] H. Munir, M. Moayyed, and K. Petersen, "Considering rigor and relevance when evaluating test driven development: A systematic review," *Information and Software Technology*, vol. 56, no. 4, pp. 375–394, 2014.

[14] Y. Rafique and V. B. Mišić, "The effects of test-driven development on external quality and productivity: A meta-analysis," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 835–856, 2012.

[15] B. Turhan, L. Layman, M. Diep, H. Erdogmus, and F. Shull, “How effective is test-driven development,” *Making Software: What Really Works, and Why We Believe It*, pp. 207–217, 2010.

[16] B. A. Kitchenham, “Systematic review in software engineering: where we are and where we should be going,” in *Proceedings of the 2nd international workshop on Evidential assessment of software technologies*, 2012, pp. 1–2.

[17] “Prisma,” 2015. [Online]. Available: http://www.prisma-statement.org/

[18] H. A. Mohamed Shaffril, S. F. Samsuddin, and A. Abu Samah, “The abc of systematic literature review: The basic methodological guidance for beginners,” *Quality & Quantity*, vol. 55, no. 4, pp. 1319–1346, 2021.

[19] C. Wohlin, “Guidelines for snowballing in systematic literature studies and a replication in software engineering,” in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.

[20] A. Hunt and D. Thomas, *Pragmatic unit testing in Java with JUnit*. The Pragmatic Bookshelf, 2003.

[21] R. Miller and C. T. Collins, “Acceptance testing,” *Proc. XPUniverse*, vol. 238, 2001.

[22] M. Hilton, J. Bell, and D. Marinov, *A Large-Scale Study of Test Coverage Evolution*. New York, NY, USA: Association for Computing Machinery, 2018, p. 53–63. [Online]. Available: https://doi.org/10.1145/3238147.3238183

[23] H. Zhu, P. A. Hall, and J. H. May, “Software unit test coverage and adequacy,” *Acm computing surveys (csur)*, vol. 29, no. 4, pp. 366–427, 1997.

[24] G. Duggal and B. Suri, “Understanding regression testing techniques,” in *Proceedings of 2nd National Conference on Challenges and Opportunities in Information Technology*, 2008.

[25] K. Beck, *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[26] A. Abran, R. E. Al-Qutaish, J.-M. Desharnais, and N. Habra, “An information model for software quality measurement with iso standards,” in *Proceedings of the International Conference on Software Development (SWDC-REK), Reykjavik, Iceland*, 2005, pp. 104–116.

[27] A. Tosun, O. Dieste, S. Vegas, D. Pfahl, K. Rungi, and N. Juristo, “Investigating the impact of development task on external quality in test-driven development: An industry experiment,” *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2438–2456, 2021.

[28] D. Fucci, S. Romano, M. T. Baldassarre, D. Caivano, G. Scanniello, B. Turhan, and N. Juristo, “A longitudinal cohort study on the retainment of test-driven development,” in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.

[29] D. Fucci, G. Scanniello, S. Romano, M. Shepperd, B. Sigweni, F. Uyaguari, B. Turhan, N. Juristo, and M. Oivo, “An external replication on the effects of test-driven development using a multi-site blind analysis approach,” in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2016, pp. 1–10.

[30] D. Fucci, B. Turhan, and M. Oivo, “Impact of process conformance on the effects of test-driven development,” in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 1–10.

[31] D. Fucci, B. Turhan, and M. Oivo, “Conformance factor in test-driven development: initial results from an enhanced replication,” in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–4.

[32] “Casp checklists - casp - critical appraisal skills programme,” Apr 2022. [Online]. Available: https://casp-uk.net/casp-tools-checklists/

[33] “Kata-log.” [Online]. Available: https://kata-log.rocks/bowling-game-kata

[34] “Kata-log.” [Online]. Available: https://kata-log.rocks/mars-rover-kata

[35] G. Goldet and J. Howick, “Understanding grade: an introduction,” *Journal of Evidence-Based Medicine*, vol. 6, no. 1, pp. 50–54, 2013.

[36] “The gii-grin-scie (ggs) conference rating,” Oct 2021. [Online]. Available: https://scie.lcc.uma.es:8443/

[37] “Systematic review reporting standards,” Nov 2017. [Online]. Available: https://www.roses-reporting.com/

[38] “Critical appraisal checklists.” [Online]. Available: https://bestpractice.bmj.com/info/toolkit/ebm-toolbox/critical-appraisal-checklists

[39] “Critical appraisal tools.” [Online]. Available: https://jbi.global/critical-appraisal-tools

[40] D. Fucci, B. Turhan, N. Juristo, O. Dieste, A. Tosun-Misirli, and M. Oivo, “Towards an operationalization of test-driven development skills: An industrial empirical study,” *Information and Software Technology*, vol. 68, pp. 82–97, 2015.

[41] S. Romano, D. Fucci, G. Scanniello, B. Turhan, and N. Juristo, “Findings from a multi-method study on test-driven development,” *Information and Software Technology*, vol. 89, pp. 64–77, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584917302550

[42] S. Hammond and D. Umphress, “Test driven development: the state of the practice,” in *Proceedings of the 50th Annual Southeast Regional Conference*, 2012, pp. 158–163.

[43] D. Fucci, H. Erdogmus, B. Turhan, M. Oivo, and N. Juristo, “A dissection of the test-driven development process: Does it really matter to test-first or to test-last?” *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 597–614, 2016.

[44] A. Čaušević, D. Sundmark, and S. Punnekkat, “Test case quality in test driven development: A study design and a pilot experiment,” in *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012).* IET, 2012, pp. 223–227.

[45] G. Scanniello, S. Romano, D. Fucci, B. Turhan, and N. Juristo, “Students' and professionals' perceptions of test-driven development: a focus group study,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 1422–1427.

[46] S. Romano, D. Fucci, G. Scanniello, B. Turhan, and N. Juristo, “Results from an ethnographically-informed study in the context of test driven development,” in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, pp. 1–10.

[47] K. Buffardi and S. H. Edwards, “Impacts of adaptive feedback on teaching test-driven development,” in *Proceeding of the 44th ACM technical symposium on Computer science education*, 2013, pp. 293–298.

[48] K. Buffardi and S. H. Edwards, “Exploring influences on student adherence to test-driven development,” in *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, 2012, pp. 105–110.

[49] F. G. Rocha, L. S. Souza, T. S. Silva, and G. Rodríguez, “Enhancing the student learning experience by adopting tdd and bdd in course projects,” in *2021 IEEE Global Engineering Education Conference (EDUCON).* IEEE, 2021, pp. 1116–1125.

[50] I. Karac, B. Turhan, and N. Juristo, “A controlled experiment with novice developers on the impact of task description granularity on software quality in test-driven development,” *IEEE Transactions on Software Engineering*, vol. 47, no. 7, pp. 1315–1330, 2019.

[51] B. K. Papis, K. Grochowski, K. Subzda, and K. Sijko, “Experimental evaluation of test-driven development with interns working on a real industrial project,” *IEEE Transactions on Software Engineering*, 2020.

A Appendix 1

Review Protocol

Administrative Information

Title:	Test-Driven Development with the Focus on Inexperienced Programmers: A Literature Review.
Authors:	Adam Nyman, Linnaeus University, an223pk@student.lnu.se Oliver Rimmi, Linnaeus University, or222cb@student.lnu.se
Supervisor:	Rüdiger Lincke , rudiger.lincke@lnu.se

Introduction

Rationale:	There seems to be a lot of research conducted within the field of Test-driven development. However, there seem to be divided opinions on its usefulness and a lack of reports that compare the result between these reports.
Objectives:	1. What are the effects on external quality when inexperienced developers use Test-driven development compared to Test-last development practices? 2. What other effects can be identified when inexperienced developers use Test-driven development compared to Test-last development practices. 3. What difficulties can an inexperienced developer encounter when adhering to the Test-driven development practices.

Figure 1.1: Administrative information and introduction section from our review protocol.

Method

Eligibility Criteria:	The focus of this review will be on articles written within the timespan of 2012-2022 and in the English language. Furthermore, all the articles in this review have a focus on inexperienced programmers. The literature review will be a qualitative study. The timeframe of the study is during the Spring term of year 2022.
Information Sources:	IEEE, ACM, Google Scholar
Search Strategy:	ACM [[**Title**: tdd] **OR** [**Title**: "test-driven development"]] **AND** [**All**: novice inexperienced student intern] **AND** [**Publication Date**: (01/01/2012 **TO** *)] [[Title: tdd] OR [Title: "test-driven development"]] AND [All: efficiency productivity] AND [All: novice inexperienced student intern] AND [Publication Date: (01/01/2012 TO *)] IEEE (("Document Title":TDD) OR ("Document Title":test-driven development)) AND (("All Metadata":novice) OR ("All Metadata":inexperienced) OR ("All Metadata":student)) GS allintitle: "test-driven development" OR TDD novice OR inexperienced OR student OR intern
Study records:	
Data management:	We had a spreadsheet where we put all of the relevant articles and their important metadata.
Selection process:	In this article, CASP will be used to filter

Figure 1.2: Method section from our review protocol.

	articles based on quality as well as bias.
Data collection process:	1) Collect articles using search queries. 2) Select papers by reading the title and abstract. If relevant we place them in a spreadsheet. 3) Filter papers by quality using CASP. Add a link in the spreadsheet to CASP documents for papers that pass the CASP check. 4) Write a summary of each article about its method and results. 5) Collect data from the summaries.
Data items:	- External Quality - TDD - Test-driven development - Inexperienced Developers - Student - Novice - Intern - Comparison
Outcomes and prioritization:	1. TDD does/"does not" have a beneficial effect on external quality for inexperienced programmers. - In this case, we figure the result of this outcome will show whether it is worth teaching inexperienced programmers TDD or not. 2. The effects of TDD are more beneficial / less beneficial compared to other practices. - In this case, the result will tell whether TDD is a good option or whether there are better alternatives for developing code. 3. Try to examine possible problems that tend to occur in development using the TDD practice. - Hopefully, this will show which skills are beneficial when using TDD.
Risk of bias in individual studies & Meta-bias(es):	To avoid using articles that are biased, we intend to grade our chosen articles using CASP checklists to see which ones might be unsuitable. This will be done at study level and the articles with a lower CASP grade will be mentioned in our data synthesis if used.
Data synthesis:	We have no quantitative values.
Confidence in cumulative evidence:	We will look at it using GRADEs five

Figure 1.3: Continuation of Method section from our review protocol.