Test-Driven Development: Drawbacks, Benefits, Industrial Usage and Complementary Methods

Freja Ekman, Sarah Johannesson, Ellen Peber, Calle Sandberg | Faculty of Engineering at Lund University
fr6744ek-s, ine15sjo, ine15epe, ine14csa (@student.lu.se)

***Abstract*—*Background*: This report examines how testing is performed in the Agile practise of test-driven development (TDD). One common misconception is that TDD only refers to test-first development, i.e. seeing the test fail before developing the source code. Instead, TDD also includes the process of short and steady development cycles. *Aim*: This report aims to critically investigate the Agile practise TDD in terms of its advantages and disadvantages. The paper is also to explore if other Agile test methods can lessen the practice's disadvantages, how common TDD is in the industry and how usage of the practise can increase. *Method*: To achieve the aim of the report a theoretical review of literature within the area Agile testing and TDD were conducted. *Conclusion*: Through analysis and discussion of data it could be concluded that: (1) Code quality can be improved by implementing TDD but it is unsure if the method improves or even lessens productivity and usability. (2) TDD can benefit from using complementary Agile test methods like Acceptance test-driven development, Behaviour-driven development and Exploratory testing. (3) TDD has a low adoption by industry but the usage might increase if risks are minimized and complementary Agile test methods are used.**

I. Introduction

A. Project background

The Agile Manifesto provides four guiding principles for how to work with software development in an agile manner. In short, the principles emphasises broadly the value of employees, working software, responding to change and customer collaboration. [1] Moreover, Agile software development is incremental in nature, regression testing focused, and includes the practise of writing tests prior to production code [2].

Agile software test processes differ from traditional test processes in many ways [3]. A clear and interesting process difference originates from the way requirements are viewed; for example, the Waterfall model aims to elicit most requirements in the early stages of a project, while Agile software development only identifies a few [2]. This has a direct impact on the way tests are viewed and conducted since test and requirements are linked, or more specifically "The tester is the one who verifies the product and ensures it fulfils the requirements you specified" [4].

The concept of testing before coding exemplifies how Agile methods impact the relationship between requirements and tests. Instead of looking at testing as an activity following requirement specification and coding, test cases are formed prior to coding. Test-driven development (TDD) is an Agile development approach that encompasses this [2]. Hence, TDD can be said to flip the traditional relationship between tests and requirements. Note that TDD is not the only method for working in an agile manner, it is only one of several [5], [6], [7]. Moreover, TDD is central in several modern agile software development processes, e.g. in Extreme Programming (XP) [2].

This report aims to, in a structured way, collect, summarize and critically examine research information from the area of Agile testing, focusing on the concept of test-driven development (TDD). The main focus is laid on TDD since the practice, according to Karac and Turhan [8], is very controversial in regards to its effect on software quality and programmer productivity.

B. Project purpose

The purpose of this project is threefold:

- To investigate the Agile practise TDD, a specific area of software development
- To critically examine and analyse TDD in an industrial context
- To suggest further research within the examined area

C. Problem statements

Considering the project purpose, the project is to answer the following three problem statements:

- What are the advantages and disadvantages of TDD in terms of quality, productivity and usability?
- How can complement usage of other Agile test methods lessen the disadvantages of TDD?
- How common is TDD in the industry and how can industry usage potentially increase?

D. Limitations

The project is subject of the following limitations:

- *Time limitation*. According to the course plan for ETSN20, the project should equal 2 weeks of full time studies (3 ECTS credit points) per student. Hence, the project has been limited to only study a limited number of publications from Lund University's online library, LubSearch, and Google Scholar.
- *Knowledge limitation*. The project is limited by the authors' knowledge of the area of Agile testing which will impact the level of the written report and to which dept the report will answer its problem statements.
- *Scope limitation*. Given the time and knowledge limitations, the scope of the report has been limited to mostly focusing on the Agile practise TDD.

E. Method

This report is built on a theoretical review of literature within the area of Agile testing and TDD. Resources were mainly derived through the use of the tools LUBsearch and Google Scholar. This since the tools are widely accepted as providers of high quality academia resources.

II. Theory

The theory of this report begins whit a section covering the Agile Manifesto (section A). Thereafter section B explains the role of testing, the relationship between requirements and tests and the agile method of using test cases as requirements. Section C then highlight the differences between agile testing and traditional testing and section D explains the concept of TDD. Lastly, section E introduces how TDD can vary in different settings.

A. The Agile manifesto

Beck et.al. [1] drafted the The Manifesto for Agile Software Development in 2001 which consist of four main principles, namely; "Individuals and interactions over processes and tools, Working software over comprehensive documentation, Customer collaboration over contract negotiation, Responding to change over following a plan". The authors of the Manifesto stresses that they do see process, tools, documentation, contract negotiation and following a plan as valuable. However, the contrast to these as mentioned in the principles are more valuable.

According to Darrin and Devereux [9]; the Agile Manifesto could be interpreted as an unstructured way of working. Whoever, the authors emphasises that in reality, the opposite applies since agile processes "... require both consensus among the team and a high level of discipline to follow and execute the agreed upon rules and methods". Moreover Darrin and Devereux mean that agile methods are especially appropriate to use in today's projects given their fast-changing, and therefore uncertain, environment.

B. Agile Software Testing

Khan, Srivastava and Pandey [3] explains that since testing assures and improves the product quality, it is an important part of any software project. The authors describe software testing as ".. a process for verification and validation of the software, it is a process to guaranty execution of error free and bugs free Software/applications" and Bjarnason and Borg [4] highlights that the tester guarantees that the specified requirements are satisfied.

Bjarnason and Borg [4] have also observed that there is a tendency of disconnect between requirements and testing in many companies, or as they put it "...requirements engineering and testing (RET) alignment is a significant challenge for many companies". Moreover, they emphasize that a common requirement understanding within a project is crucial for RET alignment. In the article three strategies to ensure RET alignment in three different scenarios are presented. The strategies are to reduce distances between key players, harvest trace links, and use test cases are requirements. The later strategy is often used by Agile developers and comes in different variants in the industry. Generally, the idea is for developers to create requirements as they go and document these as test cases and thereby reducing the total documentation effort (avoid traditional requirements specification). Furthermore, Khan, Srivastava and Pandey [3] states that software can benefit from constructing test before code.

The idea of testing before coding is a part (not the entirety as often misconceived) of the practice called test-driven development (TDD). TDD also involves refactoring, focusing on little tasks and more. [8] TDD is more thoroughly described in the coming section D.

C. Agile testing vs. traditional testing

Khan, Srivastava and Pandey [3] describes that Agile software testing methodologies or processes differs from traditional testing in five ways. Firstly, they are incremental in nature rather than sequential and for each iteration thorough testing is performed so that that the next iteration can start without issues. Secondly, people are in focus rather than tools and process. For testers, this means that they to a higher degree coordinate with developers and customers. Thereby they are knowledgeable of the requirements, ensuring a clear link between these and the system. Thirdly, test documentation is less valuable than live software. Extensive documentation is avoided through teamwork and face to face communication within the team and with the customer. Fourthly, collaboration and direct customer feedback is valued over contract negotiation. Lastly, flexibility is prioritized over planning and accommodating to changes are worth more than following the plan.

D. Test-driven development

As visualized in Figure 1, test-driven development (TDD) is used in the agile method Extreme Programming (XP) [10] but can be carried out as a stand-alone agile method itself [11]. Since the methodology is to write unit tests for the new functionality that will be added, see the test fail, write just enough code to make the test pass and lastly refactor the code but also re-run the tests [10] it is easy to assume that the only purpose of TDD is to write tests before coding. This is not the case according to Karac and Turhan [8] who claims that it also relates to short and steady development cycles corresponding their duration and refactoring effort. Additionally, they inform that the TDD process should be adjusted as needed which also is on the same discourse as the agile manifesto that states that one should respond to change over following a plan [12].

Two another articles by Fucci et al. and Tosun et al. tells that the concepts of testing first or last does not have any impact on the quality or productivity. Also, refactoring, that are a part of TDD, is revealed to have a negatively outcome on both quality and productivity. However, the process of TDD which encourage steady, fine-grained, small-scale steps improves the development when it comes to focus and flow. [13], [14]

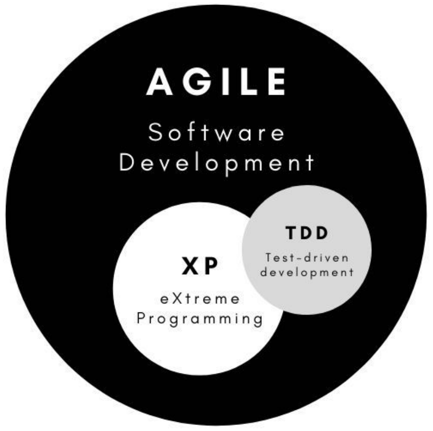

Fig. 1. Visualisation of the relation between Agile software development, XP and TDD (XP is an Agile method incorporating TDD but the Agile practise TDD can also be used as a stand-alone method).

E. Variants of Test Cases as Requirements

To document requirements as test cases is also known as using test cases as requirements (TCR) and it is a practise used to increase alignment between requirements and development activities as e.g. testing [15].

An iterative case study by Bjarnason et. al. [15] of three companies identified five different use-types/variants of the test-cases as requirements (TCR) practise: stand-alone strict, stand-along manual, de facto, behavior-driven and story-test driven. These where identified using four facets of TCR observed in the companies, namely; documentation time frame for defining TCRs (upfront / during elicitation and validation / after-the-fact during the testing process), requirements format for the TCRs (ranges from formal domain-specific language structure to natural language test cases), machine executable specification (yes/no/partly) and TCR-specific tool support (yes/no). Table I shows how the four facets are combined to characterise the TCR use-types. In this report TCR is regarded as a sub-part of TDD.

The authors describe the variants in the following way: In stand-alone strict TCR only a part of the functionality are specified with automated test cases whereas in stand-alone manual TCR stakeholders using manual test cases result in specified requirements. De facto uses manual and automatic test cases to yield the requirements and in behaviour-driven TCR requirements are specified in the elicitation process by the use of automated acceptance test cases. Lastly, in story-test driven TCR user stories and acceptance criteria are used for requirement specification and manual/automatic test cases.

Moreover, the authors findings exemplifies how the TCR practise can be adapted to company specific conditions and thereby vary between companies. The studied companies' TCR practise also varied in terms of in which context it was used, in which extent and to what degree of maturity. Lastly, the authors highlight that more variants of TCR can potentially be identified by examining more companies and altering the facets.

	Stand-alone strict	**Stand-along manual**	**De facto**	**Behavior-driven**	**Story-test drive**
Documentation time	Upfront	Upfront	After-the-fact	Upfront	Upfront
Requirements format	Semi-structured	General	General	Structured	Semi-structured
Machine executable specification	Yes	No	Partly	Yes	Partly
TCR-specific tool support	Yes	No	No	Yes	Yes

TABLE I
VARIANTS OF THE PRACTISE TEST-CASES AS REQUIREMENTS (TCR) CHARACTERISED BY FOUR DIFFERENT FACETS

III. Analysis

Section A of this analysis will broadly introduce advantages and disadvantages of TDD. The following sections (B, C and D) will go in to depth in discussing advantages and disadvantages of TDD when it comes to code quality, productivity and usability. Section E will then examine how other methods for Agile testing compare to TDD and section F will discuss how complementary usage of other test methods can minimize the disadvantages of TSS. Lastly, section G will discuss what further research is needed within the area.

A. Advantages and Disadvantages of TDD

When it comes to the advantages of employing TDD, it is claimed to bring improvements in code quality, productivity and defect intensity as a few examples [16], [17]. At first glance, TDD might seem like an optimal solution with no drawbacks. However, according to 48 different empirical studies on TDD it is not that easy to implement. This has several reasons; increased development time, developers being unable to adapt and technical limitations. [18]

According to an article by Karac and Turhan the method was not widely adopted in the industry at the time, in 2018, and critics wanted more evidence to certify the assets of TDD [8]. Previous research studies, that had been investigating the effectiveness of the method, had failed to produce conclusive results regarding the improvements obtained with TDD. In fact, all possible outcomes (positive, neutral and negative) were reported. [16]

B. Code Quality

Almost all case studies, when it comes to improvement in code quality, were found to report an improvement [16][17]. The quality improvement gained with TDD, by continuous and rigorous testing and refactoring lead to a lower defect density [19]. Beside this, also a faster rate of defect detection and fixing, as well to an easier and faster maintenance [16].

In other words, TDD facilities modular code that is easier for programmers to understand and debug. It can, therefore, eventually help programmers to earlier detect defects and fix them. At the same time, tests can act as documentation and improve the understandability of the code. Lastly, TDD also supports the programmers to think from the perspective of an end user.[19]

C. Productivity

Research display that effects on productivity are inconclusive, but most results seem to show a slight decrease in productivity as a result of increased development time [20].

A hypothesis, according to Khanam, Zeba and Ahsan, is that in some situations unit testing becomes more time consuming and inappropriate than effective. This since, not all real-life systems can be generalized and simplified for unit testing. A forced unit testing may lead to increased time in rectifying the unit tests, than developing the code and performing functional testing [19].

While others, suggested that improvement in code quality by TDD (leading to lower defect generation and faster fixes), thereby improved the productivity of the developers as well [16].

D. Usability: Human errors and Technical limitations

As previously mentioned TDD is not commonly used in the industrial environment. In a research following 416 developers claiming to use TDD, only 12% did this in a correct matter [8]. Thus, there is a difficulty in following TDD-protocols. This is a result of lack of knowledge in the area and that people have different perception of what TDD actually brings to the table. Additionally, developers who lack a skill in testing seem to have difficulties adapting to this technique. These are not a flaw of the method itself, but indicates that there is a complexity to TDD that makes it hard to pick up in any project [18].

According to a study by Causevic, Sundmark and Punnekkat there are also more technical obstacles when implementing TDD; limited tool support, lack of front-level design and inability to adjust legacy code (the existing codebase) accordingly. All of these make it difficult to use TDD in projects where others methods were previously used, as it is hard to adjust the existing work and new tool support is needed. Development might have to be redone completely. Also, using TDD generates a constantly changing design that could make further development more difficult [18].

E. TDD compared to other agile testing methods

How does other methods for Agile Testing compare to TDD? Many of the advantages from using TDD can be achieved with other methods that include short development cycles. This since the generally misconceived "core" of TDD, that tests are written before the code, might not actually make much of a positive difference for a software development project [8]. As described earlier the actual benefited effect of TDD is not the test-first approach but the structured cycles.

In this report, three other testing methods within the agile approach will be discussed to research if they could be a better alternative to test-driven development. The evaluated methods are behaviour-driven development, acceptance test-driven development and exploratory testing.

1) Acceptance test-driven development: Acceptance test-driven development (ATDD) is an approach to discover requirements where the acceptance tests are created by the customer [5]. Therefore ATDD is not just about how to create tests but also to clarify the requirements [21]. Furthermore, there are results of using ATDD that shows a double increase in productivity. This may vary from team to team but a number of them experience the productivity increase and also improvements concerning quality. However, both TDD and ATDD has the same quality goal and the acceptance tests can form which unit tests that are needed to develop. [5]

2) Behaviour-driven development: Behaviour-driven development (BDD) is a direct response to the issues that are found in TDD and therefore it is also based on TDD but also ATDD. The purpose of BDD is to have a language that makes it easier to define the wanted behaviour of the system that generates clear executable specifications, i.e tests. The language that is used is actually the core of BDD and it should be ubiquitous which means that there is a structure based on the domain model. Once there is a common language it is easier to communicate with the customer and about the solutions. On this basis, stories and scenarios are written to make the need for the system clear and to achieve features and tests. These scenarios will be the ground to confirm if acceptance tests are passed. To pass the tests, the same process as in TDD is also required, namely the steps of failing, pass and refactoring. Additionally, since the terminology is crucial in BDD, the coding should follow the ubiquitous language to make it easier to understand the code and to minimize the additional documentation. Moreover, the language evolves during different phases and it is of course approved and needed to update the ubiquitous language during the time. [6]

3) Exploratory testing: Exploratory testing (ET) is described as a more free test style where the design, execution and learning of tests are done at the same time to obtain continuous optimization testing. The methodology is not limited to a certain method and can be used as a compliment in combination with other test methods. Another thing that differs a lot from the other methods is that ET is not automated and scripted but instead used as manual testing to ensure that the actual using of the system is tested. Therefore it is highly recommended to use when testing things like the GUI of a system. [7]

4) The test quadrant: There is a model called the test quadrant that shows the relation between the guidance of development and the critique of the product but also the business and technology facings [22]. To cover most of the quadrant, and therefore also have greater test coverage, there is a need for a combination of the different testing techniques.

IV. Discussion

A. What are the advantages and disadvantages of TDD in terms of quality, productivity and usability?

From the section Analysis, it becomes clear that the opinions of advantages and disadvantages of TDD are divided. Different viewpoints were especially identified regarding the impact TDD has on productivity. The majority claim TDD has a negligible effect on productivity. At the same time, others sources confirm the opposite. It can be worth to point out that one finding from reviewing different studies is that they refer to productivity differently. Some measure it as the amount of time spent to accomplish a task, while others mean that it represents the quality of the work performed. It can therefore be questionable if the productivity really is negligible or if it depends on how productivity is measured? However, when it comes to code quality the common opinion is that it improves with the use of TDD.

In the terms of adoption for industry, there exist both technical obstacles when implementing, an using, TDD, as well as different perceptions of what it is. Thus, the practice's use decreases which can be seen as a drawback with TDD. As a result of this, it does not come as a surprise that only 12% use TDD in a correct way according to the study mentioned in the analysis by Karac and Turhan. This drawback could perhaps be minimized by implementing better guidelines of how to use TDD and by increasing the industry knowledge of the practise. Another possible consequence of the inadequate agreed perception of TDD is that many of the studies on TDD are contradictory.

Lastly, it is worth to point out that the overall advantages with TDD is not due to the test-first dynamic. It is rather a result of the structural cycles used in TDD.

B. How can complement usage of other Agile test methods lessen the disadvantages of TDD?

As seen in the analysis each testing technique has its own expertise. ATDD focuses on the close relationship with the customer to provide clear requirements while BDD attaches great importance to the language and lastly, ET targets manual testing. None of these methods is mutually exclusive and could be used together to achieve the greatest advantages.

The uncertainty of how TDD should be used could be cleared by using BDD so that the whole team is following the same guideline. Additionally, ATDD will make the purpose of the system unambiguous and there will not be as many questions as when using TDD. The risk of TDD not resulting in higher productivity, which has been observed out in the industry, can be lowered by implementing ATDD which has a clearer, or proved, productivity benefit. Also, ET will be a strong complement since TDD does not cover manual testing but instead unit tests.

Moreover, the model called the test quadrant could motivate how the different testing methods should be used. As described, if the whole quadrant is filled in it gives a larger code coverage. Thus, complementing TDD with other Agile Testing processes will most probably result in better feedback to the developers. Furthermore, this is an assurance of that the right product is developed and a decreasing uncertainty of the process.

C. How common is TDD in the industry and how can industry usage potentially increase?

The difficulties with TDD, that are mentioned in the analysis, can be summarized to 7 reasons why the method has been adapted to a limited extent in industrial environments. These are increased development time, insufficient TDD experience and knowledge, insufficient design, insufficient developer testing skills, insufficient adherence to the TDD protocol, domain- and tool-specific limitations and legacy code. As stated in the analysis, TDD is not as widely used in the industry, which could be the result of these limitations. To potentially be able to increase the usage of the method there must be strategies to counter them.

The insufficient knowledge of TDD, adherence to the TDD protocol and testing skills amongst developers could be handled by better educating the people involved. However, the struggle probably occurs not only due to lack of knowledge but due to people being used to work in other ways and not being willing to adapt. If this is the case a bigger cultural change in the organisation would be needed, where the positive attitude towards continuous learning and knowledge of several development methods should be encouraged.

To handle the problem of domain- and tool-specific limitations relevant support has to be implemented. A decision to use TDD as the way of working has to be taken by managers with a high enough authority for investment and organisational adaption to follow. Without authority, the decision will not result in actual action and the TDD practise is at risk of not getting enough support. Ways to handle legacy code should also be included in this adaption. Exactly which changes are needed depends on the specific situation.

The increased development time and lack of sufficient design are the last problems to manage. However, these are more difficult to counter as they are is some ways results of the benefits of TDD. Use of the agile and adaptive method TDD generates code of better quality, and a process that can handle changes, but results in more time being spent developing and a looser framework to follow. If the benefits out-way the drawbacks in a specific situation, this should be argument enough to adapt to this way of working. Whether the benefits out-way the drawbacks depend on the company and its requirement, for example; a nuclear power plant application might value design/structure more than being change-adaptive, while a music application might value the opposite.

Even though the transition from a more static method to TDD may not be smooth, there will be a payoff for committing if risks are handled. Also, companies must be ready to adapt the practise after company specific needs and conditions and create their own version of TDD. Companies might even, as mentioned in the previous section, complement TDD with other practises/methods in order to minimize the practice's risks.

D. Further research

The area of TDD could be further explored to make it easier for academics and companies to understand the practice's advantages and disadvantages. For instance, more case studies would increase the information of how the method is used today and enable academia to suggest clearer guidelines for how the technique is to be performed in a successful manner. Also, the guideline could include how to move from a traditional testing process to TDD and the risks within each step of the transition, which could increase the usage in the industry. Another option to this guide is to include instructions on how to use TDD with complementing techniques to increase the proven benefits of the work process.

It would also be interesting to discover or develop a good tool support to use TDD. This could maybe, somewhat, solve the problem with inconsistent usage of TDD by supporting the developer through the different process steps.

V. Conclusion

A. What are the advantages and disadvantages of TDD in terms of quality, productivity and usability?

Generally, the advantages of TDD is more a result of the structural cycles used in TDD, rather than the distinctive test-first dynamic. Moreover, code quality is primarily positively affected by the use of TDD. The effects on productivity and usability from using TDD is on the other hand inconclusive and perhaps even negligible. This might be a result of the inadequate agreed perception of productivity and TDD.

B. How can complement usage of other Agile test methods lessen the disadvantages of TDD?

By applying other Agile test methods like ATDD, BDD and ET that are discussed in this paper, TDD can be complemented and also the disadvantages of TDD will lessen. Some of the methods are even based on TDD and are designed to increase the practice's benefits. ATTD will answer questions of uncertainty, BBD ensures that the whole team is following the same guideline and ET is covering manual testing which is excluded in TDD.

C. How common is TDD in the industry and how can industry usage potentially increase?

TDD is not widely used in the industry as a result of several limitations. To minimize the affect of these limitations, and eventually increase usage of the method, they must be countered. This could be performed by; making investments in, and taking actions to, handle technical limitations, educating staff members in TDD and testing and by complementing TDD with other Agile test methods. This will also make the transition to TDD easier.

VI. Contribution statement

A. Freja Ekman

1) Responsibility: Content/scope - 'have we answered all questions and project goals?'

2) Section: Theory, analysis, discussion, conclusion

B. Sarah Johannesson

1) Responsibility: Project leader, handles contact with supervisor, schedules meetings and keeps track of deadlines, hand-ins

2) Section: Analysis, discussion, conclusion

C. Ellen Peber

1) Responsibility: Structure, layout, references

2) Section: Introduction, theory, discussion

D. Calle Sandberg

1) Responsibility: Adherence project guidelines

2) Section: Analysis, discussion, conclusion

E. Everyone

All authors was a part of the literature research, discussed the results and reviewed and edited the paper to increase the quality. The work with the report has been conducted in an iterative way. Hence the sections which the authors have been involved in should be interpreted with some caution and seen mostly as a division of responsibility. All authors have reviewed and edited each others work.

References

[1] Beck *et al.* (2001) Manifesto for agile software development. [Online]. Available: http://agilemanifesto.org

[2] Naik, Tripathy, and Wiley InterScience (Online, *Software testing and quality assurance. [Elektronisk resurs] theory and practice.* Wiley, 2008. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=cat07147a&AN=lub.6126887&site=eds-live&scope=site

[3] Khan, Srivastava, and Pandey, "Agile approach for software testing process," in *2016 International Conference System Modeling Advancement in Research Trends (SMART)*, Nov 2016, pp. 3–6.

[4] Bjarnason and Borg, "Aligning requirements and testing: Working together toward the same goal," *IEEE Software*, vol. 34, no. 1, pp. 20–23, Jan 2017.

[5] Pugh, *Lean-Agile Acceptance Test-Driven Development: Better Software Trough Collaboration*, 2011.

[6] Solis and Wang, "A study of the characteristics of behaviour driven development," in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, Aug 2011, pp. 383–387.

[7] Yu, "Design and application on agile software exploratory testing model," in *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, May 2018, pp. 2082–2088.

[8] Karac and Turhan, "What do we (really) know about test-driven development?" *IEEE Software*, vol. 35, no. 4, pp. 81–85, July 2018.

[9] Darrin and Devereux, "The agile manifesto, design thinking and systems engineering," in *2017 Annual IEEE International Systems Conference (SysCon)*, April 2017, pp. 1–5.

[10] Fucci *et al.*, "Towards an operationalization of test-driven development skills: An industrial empirical study." *Information and Software Technology*, vol. 68, 2015. [Online]. Available: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0950584915001469&site=eds-live&scope=site

[11] Beck, *Test-driven Development: By Example*, 2003, vol. 2, p. 204.

[12] Darrin and Devereux, “The agile manifesto, design thinking and systems engineering,” in *2017 Annual IEEE International Systems Conference (SysCon)*, April 2017.

[13] Fucci *et al.*, “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, July 2017.

[14] Tosun *et al.*, “Investigating the impact of development task on external quality in test-driven development: An industry experiment,” *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[15] Bjarnason *et al.*, “A multi-case study of agile requirements engineering and the use of test cases as requirements,” *Information and Software Technology*, vol. 77, pp. 61 – 79, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584916300544

[16] Rafique and 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, June 2013.

[17] Tosun *et al.*, “Investigating the impact of development task on external quality in test-driven development: An industry experiment,” *IEEE Transactions on Software Engineering*, vol. PP, pp. 1–1, 10 2019.

[18] Causevic, Sundmark, and Punnekkat, “Factors limiting industrial adoption of test driven development: A systematic review,” in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, March 2011, pp. 337–346.

[19] Khanam and Ahsan, “Evaluating the effectiveness of test driven development: Advantages and pitfalls,” *International Journal of Applied Engineering Research*, vol. 12, pp. 7705–7716, 01 2017.

[20] Amrit and Meijberg, “Effectiveness of test-driven development and continuous integration: A case study,” *IT Professional*, vol. 20, no. 1, pp. 27–35, January 2018.

[21] Hoffmann *et al.*, “Applying acceptance test driven development to a problem based learning academic real-time system,” in *2014 11th International Conference on Information Technology: New Generations*, April 2014, pp. 3–8.

[22] Gupta, Manikreddy, and GV, “Challenges in adapting agile testing in a legacy product,” in *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, Aug 2016, pp. 104–108.