Complex Systems Informatics and Modeling Quarterly (CSIMQ)
eISSN: 2255-9922
Published online by RTU Press, https://csimq-journals.rtu.lv
Article 174, Issue 31, June/July 2022, Pages 18–28
https://doi.org/10.7250/csimq.2022-31.02

A Literature Review on the Challenges of Applying Test-Driven Development in Software Engineering

Daniel Staegemann*, Matthias Volk, Maneendra Perera,
Christian Haertel, Matthias Pohl, Christian Daase, and Klaus Turowski

Otto von Guericke University, Universitätsplatz 2, Magdeburg, 39106, Germany

{daniel.staegemann, matthias.volk, christian.haertel, matthias.pohl,
christian.daase, klaus.turowski}@ovgu.de, hetti.perera@st.ovgu.de

Abstract. Due to the ongoing trend of digitalization, the importance of software for today's society is continuously increasing. Naturally, there is also a huge interest in improving its quality, which led to a highly active research community dedicated to this aim. Consequently, a plethora of propositions, tools, and methods emerged from the corresponding efforts. One of the approaches that have become highly prominent is the concept of test-driven development (TDD) that increases the quality of created software by restructuring the development process. However, such a big change to the followed procedures is usually also accompanied by major challenges that pose a risk for the achievement of the set targets. In order to find ways to overcome them, or at least to mitigate their impact, it is necessary to identify them and to subsequently raise awareness. Furthermore, since the effect of TDD on productivity and quality is already extensively researched, this work focuses only on issues besides these aspects. For this purpose, a literature review is presented that focuses on the challenges of TDD. In doing so, challenges that can be attributed to the three categories of people, software, and process are identified and potential avenues for future research are discussed.
Keywords: Test-Driven Development, TDD, Testing, Software Engineering, Literature Review, Quality Assurance.

1 Introduction

Due to the ongoing trend of digitalization, the importance of software for today's society is continuously increasing [1]. Naturally, there is also a huge interest in improving its quality. This can be achieved by enhancing the tools and methods used in software engineering as well as by creating new ones. Consequently, there is a highly active research community dedicated to this task [2].

* Corresponding author

Reference: D. Staegemann, M. Volk, M. Perera, Chr. Haertel, M. Pohl, Chr. Daase, and K. Turowski, "A Literature Review on the Challenges of Applying Test Driven Development in Software Engineering," Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 31, pp. 18–28, 2022. Available: https://doi.org/10.7250/csimq.2022-31.02

Additional information. Author ORCID iD: D. Staegemann – https://orcid.org/0000-0001-9957-1003, M. Volk – https://orcid.org/0000-0002-4835-919X, and M. Pohl – https://orcid.org/0000-0002-6241-7675. PII S225599222200174X. Received: 15 June 2022. Revised: 21 July 2022. Accepted: 21 July 2022. Available online: 29 July 2022.

While this led to a plethora of propositions, one of the approaches that have become highly prominent and will also be the subject of this article is the concept of test-driven development (TDD). Instead of writing the productive code and subsequently testing it to assure its correctness, as it is practiced in the traditionally applied test last development (TLD) approach, the tests are written at first and the functionality is implemented afterwards [3]. This results in changes to the software design and higher test coverage, which in turn lead to an improvement in the quality of the developed application [4] but may result in lower initial productivity compared to TLD [5]. Those effects are especially noticeable in an industrial setting, which might be caused by larger tasks and higher developer experience in contrast to academic settings [6].

However, such a big change to the followed procedures is usually also accompanied by major challenges that pose a risk to the achievement of the set targets, as opposed to minor ones that might just result in an inconvenience. To find ways to overcome the challenges, or at least mitigate their impact, it is necessary to identify them and subsequently raise awareness. For this purpose, here, a literature review is presented that focuses on the challenges of TDD. Therefore, the research question discussed is as follows.

RQ: Which are the major challenges when applying test-driven development in software engineering?

The results can act as a foundation for succeeding research endeavors that can build upon these findings in the pursuit of determining ways to facilitate the application of TDD as well as to increase its effectiveness. Furthermore, the explicit limitation to software engineering is due to the fact that TDD also finds application in other domains such as ontology development [7], [8] and process modeling [9]. However, these areas are out of the scope of this work and might, potentially, be accompanied by different challenges.

The remainder of this work is structured as follows. After the introduction, the most relevant concepts are briefly discussed in the background (Section 2), laying the foundation of a common understanding. Afterwards, the review protocol for the literature review is outlined in Section 3. This is succeeded, in Section 4, by a presentation of the papers that have been retrieved through the search process. Their insights are aggregated and discussed in the ensuing Section 5. Finally, a conclusion is given in Section 6, recapitulating the work, highlighting possible weaknesses of the study, and proposing avenues for potential future research endeavors.

2 Background

As already mentioned in the introduction, previous research outlined that the application of TDD generally leads to an improved quality of the developed application [4]. This is primarily based on two aspects. On the one hand, through the increase of the test coverage that is associated with the use of TDD, the likelihood to detect errors is improved. On the other hand, the application of TDD also influences the design of the developed system by facilitating its decomposition into smaller parts. In doing so, the complexity for the developers is reduced, which, in turn, helps to avoid errors and increases the maintainability [10], [11].

In the traditional TLD approach, features are conceptualized, implemented, and then tested. When applying TDD, this order is changed. While the first step mostly remains identical, stronger emphasis is put on decomposing the desired functionality into small, capsulated portions [12]. This is followed by the writing of the tests. To make sure that these actually test new aspects, they are then executed. Since the actual implementation has not taken place until this point, they are expected to fail [3]. However, if the tests are passed, this shows that they are not covering a new functionality. Therefore, it would be necessary to rework them until they do. When the tests for the current iteration are ready, the actual implementation work to enable the desired capability can be performed. At this stage, aspects that go beyond the pure functionality, such as the code's elegance or how well it adheres to conventions (e.g., naming or style) can be ignored, as long as the tests provide a positive result [10]. Once this has been achieved, a refactoring phase ensues to enhance the overall quality of the code [3]. Here, the style and performance (e.g., runtime or

memory consumption) of the implementation are improved without altering the resulting functionality. This is facilitated by the earlier written tests that are used to detect if errors were introduced during the refactoring.

As stated earlier, this procedure, which has a heavy focus on incremental changes and small tasks [13], generally increases the test coverage and allows for shorter test cycles [14]. Moreover, by encouraging the developers to break down the system into many smaller pieces and separate components, TDD also heavily influences the resulting internal design [15].

While unit tests generally constitute the backbone of TDD, they are typically supposed to be augmented with other types of tests. In this regard, for instance, system or integration tests [16] are noteworthy. Especially the former are considered to be essential [17].

Furthermore, since some of the main aspects of TDD are the short test cycles and a high test frequency, running the tests manually is usually not feasible or at least not desirable. This would divert valuable time and attention from the developers that could be used more beneficially for the advancement of the desired application instead of spending it on monotonous routine tasks that can, nevertheless, still be error-prone. Therefore, test automation plays an important role. To facilitate it, the utilization of continuous integration (CI) pipelines for the test execution is common [18], [19]. This way, the existing tests can be automatically run by a CI server once a change to the code is made, and it is thereby checked if any errors have been introduced.

3 The Review Protocol

To obtain the desired insights into the challenges that accompany the application of TDD, a structured search was conducted using Scopus. Since it is arguably the largest abstract and citation database for scientific literature and covers the contributions of numerous reputable research outlets, its results promise a comprehensive overview that allows to answer the RQ.

The review follows the recommendations of Levy and Ellis [20] and Webster and Watson [21] to ensure a high degree of comprehensiveness and comprehensibility [22]. Accordingly, this resulted in a multi-stepped procedure that is further described in the remainder of this section.

To find the relevant papers, the search string was composed as follows:

"test driven development" OR "test-driven development" OR "test driven design" OR "test-driven design" OR "test first design" OR "test-first design" OR "test first programming" OR "test-first programming".

The above search string was applied to the title, assuring a high relevancy of the found publications to the TDD domain. Yet, by not already addressing the challenges and instead, initially, broadly looking for papers that deal with TDD, the scope was deliberately widened to avoid excluding entries that might be of interest for answering the RQ, despite not actually reflecting it in the title.

As further conditions for inclusion, contributions had to be written in English and published in conference proceedings or a journal to have the corresponding peer review process as a quality gate. Moreover, the subject area had to be stated as computer science. This was used to narrow the search and retrieve only the relevant articles instead of ones from unrelated domains.

Using those parameters, initially, 262 contributions were found. However, not all articles were suitable for the researched topic. Therefore, additional filtering was needed to reduce the list.

As the first step, the article's title was read, and some were removed since they were not applicable because the main focus was deviating from the main subject. For instance, papers like *Towards test-driven development for FPGA-based modules across abstraction levels* [23], *Why research on test-driven development is inconclusive*" [24], and *The Perception of Test-driven Development in Computer Science – Outline for a Structured Literature Review* [25] were filtered out as the main focus was, for instance, on acceptance testing or just proposing how to conduct a literature review. After this stage of the filter process, there were 127 candidates left.

In the next step, the abstract of the articles was examined to identify the most suitable items for the purpose. Articles whose primary research area was not TDD and whose research questions

were not aligned with the topics covered in this literature review were dropped. Furthermore, the same applied to contributions that only discussed the productivity as a challenge, since TDD's impact on quality and productivity has already been extensively and comprehensively covered [26] and a repetition would not yield any value. As a result, 57 articles were left to be reviewed as potentially relevant. In the following, these were read in detail and only the articles that were focused on TDD and provided insights into its challenges as (part of) the contribution were included. This led to a remaining set of ten papers that will be presented in the following section. An overview of the applied inclusion and exclusion criteria for the search process is given in Table 1.

Table 1. Inclusion and exclusion criteria

Inclusion criteria	**Exclusion criteria**
Published in conference paper or journal article	Not written in English
Is peer-reviewed	Content is not relevant to research question
Belongs to the subject area "computer science"	Discusses challenges of TDD only in the background section
Main research area is TDD	
Provides insights into the challenges of TDD	
Deals with challenges besides the productivity	

While the number of rejected contributions in the last step appears high, it is caused by the initial search term being rather open. However, papers that regard the practice of TDD itself could also contain relevant information concerning the challenges, despite it not being heavily advertised in the title or abstract. Therefore, many of these articles were only excluded after reading them in their entirety. The complete paper selection process is depicted in Figure 1.

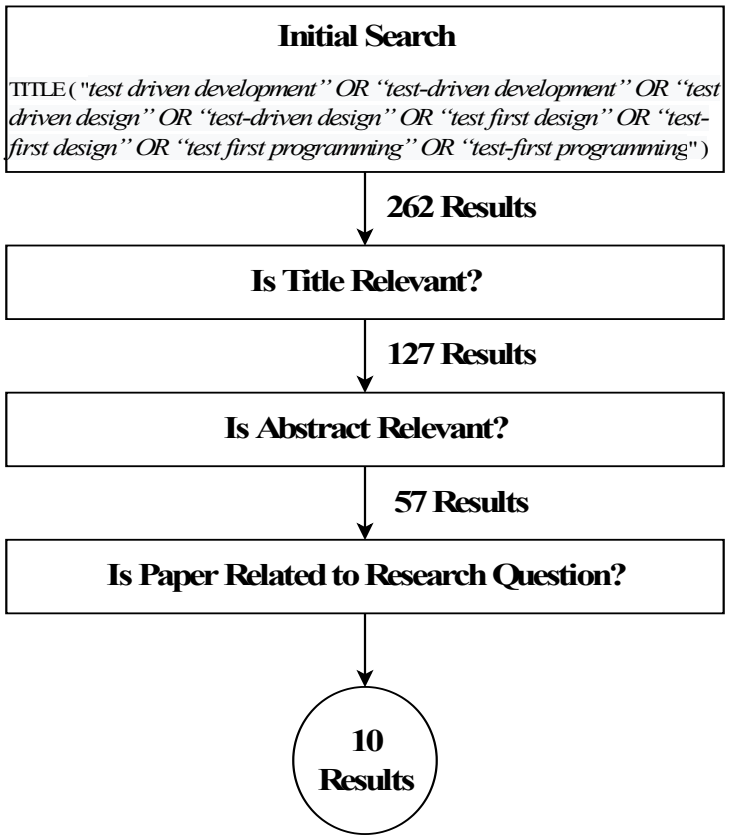

Figure 1. The paper selection process

4 Findings

As an outcome of the conducted literature review, ten publications that reveal limitations and challenges of TDD in software engineering were identified. Some were retrieved based on practical TDD applications [27], [28]. Others were derived from summarizing the past literature [29].

One noticeable aspect is the distribution by year, since there were three papers published in 2011, while the other seven contributions were spread over seven different years, as can be seen in Table 2. Further, it can be seen that the first paper that was included stems from the year 2009, despite the first publications being found already in 2002. Since the results of the initial search are covering 21 years, this means that in the first third of this period, the challenges sought for in this article have apparently been completely neglected. This is probably due to the use of the TDD approach being rather novel, leading to other aspects being focused more. From the second period, seven papers have been included, making it the time with the highest activity. Afterwards, the interest seems to have decreased again, leading to only three contributions. However, it is not clear what caused this decline. In contrast, the number of publications found in the initial search has way less fluctuation. Here, it can be seen that the first third had less activity than the second one, due to the earliest years and despite the years 2006 and 2007 being extremely productive. The last third is the one with the least amount of publications, which can, however, be attributed to the year 2022 being still ongoing. When only the years from 2008 to 2021 are regarded, there were 13, 14 publications on average, with twelve and fourteen being the most frequent numbers and only few outliers, which signifies a relatively constant interest in the topic as a whole.

In the following, the publications that were included in the final set are discussed to give an overview of the relevant literature and provide subsequent researchers and practitioners with orientation on which papers might be the most applicable ones for their own endeavors.

Table 2. Distribution of the found publications by year

Year	Found in Initial Search	Included in Final Set	References (for Publications Included in the Final Set Only)
2002	3	0	
2003	12	0	
2004	6	0	
2005	6	0	
2006	20	0	
2007	27	0	
2008	11	0	
2009	18	1	[28]
2010	14	0	
2011	11	3	[27], [30], [31]
2012	12	1	[29]
2013	7	1	[32]
2014	22	1	[33]
2015	12	0	
2016	14	0	
2017	15	1	[34]
2018	10	1	[35]
2019	12	0	
2020	12	0	
2021	14	0	
2022	4	1	[36]
Total	262	10	

Marchenko et al. [28] presented four real challenges they encountered during a three-year-long TDD project in a team at Nokia Siemens Networks. These were collected through a rigorous

interview process conducted with the team members. For them, strict regulation was needed to maintain the TDD practice. They also highlighted the application in the development of graphical user interfaces. Moreover, configuration-related development such as creating XML files was challenging due to technical complexity. Furthermore, it was difficult for them to apply TDD in their legacy code due to a lack of understanding of the requirements.

Buchan. et al. [27] discovered another interesting point during a three-year-long TDD project they conducted. They found that the top-level decision-makers of the company were unhappy with spending a considerable time creating test cases. Their impression was that more time is spent on the testing rather than working on the actual production code, which is however a misconception.

Causevic et al. [30] compiled seven challenges for adapting TDD in the industry by the means of a systematic literature review. According to the authors, the barriers are an increased development time, a lack of theoretical or practical experience in using TDD, as well as domain- and tool-specific limitations. Further, they listed the absence of a detailed design at the beginning of the project as well as insufficient skills in developing efficient and effective automated tests. Finally, insufficient adherence to the TDD practices and guidelines, and the presence of legacy code complete the list. Regarding the last challenge, the fact that TDD does not account for legacy code, and it is instead assumed that everything is developed from scratch was highlighted, which is, however, seldom a realistic assumption. This limitation has created a massive barrier for the organizations with legacy codebases and hindered adopting the process.

Sami Kollanus [31] also listed some of the literature's potential challenges, like a lack of a detailed design that causes heavy refactoring and maintenance, insufficient skill levels to apply TDD in complex tasks, and not having the correct knowledge or mindset to shift to a new paradigm. Furthermore, he revealed that the move towards this new process has a high learning curve and management support is essential for continuing the process. Apart from the issues raised, he also pointed out that the application of TDD in complex scenarios and circumstances with limited automation tools is more time-consuming, since a high volume of test code causes increased maintenance efforts and troubles.

Hammond and Umphress [29] aimed to give an overview of the state of TDD at the time of publication. Moreover, they also described several TDD extensions such as agile specification-driven development, behavior-driven design, and acceptance test-driven development. The major challenges highlighted were related to the design as well as to questions on how to conduct and structure the development.

Causevic et al. [32] amended their previous insights (see above) by the proposition of a process flow that is aimed at increasing the defect detection since, oftentimes, tests in TDD are rather geared towards providing confidence in the developed solution instead of actively looking for issues.

Roberto Latorre [33] discussed a successful application of TDD in an industrial use case. In the project, unit tests and acceptance tests have been utilized. Initially, it was not intended to work test-driven but a lack of comprehensive specifications in conjunction with a tight schedule led to the abandonment of a more waterfall-like practice in favor of TDD. This was done even though the developers were rather inexperienced with TDD, rendering this decision a risk even though all the other characteristics of the endeavor pointed to TDD being the best choice.

Nanthaamornphong and Carver [34] presented the challenges they discovered when applying TDD in developing scientific software by gathering details from the scientific community. For this purpose, they developed a survey and distributed it among 300 developers who had experience in TDD. In doing so, they found four main challenges, namely the increase in time consumption to develop tests, the difficulty in writing test cases for complex functions, writing tests for those functions where the results are still unknown since they are still in the research phase, and the necessity to spend additional time to adapt to the TDD practice due to a lack of skills and experience. Moreover, the incidental need for setting up a new environment for using TDD in their projects and the lack of tools capable of creating tests presented additional obstacles.

Karac and Turhan [35] provided a rather general overview on the topic of TDD. They highlighted, inter alia, that only a fraction of the projects that claimed to be conducted test-driven actually had the developers consequently following the corresponding methodology. In addition, the misconception, that the test first approach is all that constitutes TDD, ignoring the underlying design philosophy, was prominent. Furthermore, the developers had a hard time following the TDD process, sometimes even unintentionally. This led to them already having the production code (and not the design) in mind when writing their tests.

Baldassarre et al. [36] focused on the affective reactions of the developers supposed to apply TDD to their projects. For this purpose, they conducted three experiments, concluding that previous experience with unit testing negatively affects the perception of TDD and the corresponding activities.

An overview of the major challenges discovered in the application of TDD, mapped to the three categories, *people*, *software*, and *process*, is given in Table 3. Hereby, the first aspect pertains to challenges related to the development team and supervisors. In contrast, best practices related to tools, frameworks used in TDD applications, and code implementation guidelines are categorized as software. Finally, challenges regarding the implementation of the actual process fall into the last category.

Table 3. The major challenges of TDD

Challenge type	Description	References
People	Lack of knowledge, experience, and competencies in applying TDD	[27], [30], [32], [33], [34], [35]
	Difficulty to shift to the TDD mindset	[28], [29], [31], [32], [36]
	Senior-level management not having a proper understanding of the TDD practice	[27], [32]
Software	Technical complexity in applying TDD in certain scenarios such as GUI development, configuration development, or complex functions	[28], [30], [32]
	Lack of suitable software tools to create tests	[31], [32], [34]
Process	Lack of detailed upfront design	[29], [31], [32], [35]
	Not having proper guidelines for using TDD for legacy code	[30]
	High test code volume	[32]
	Tests are often geared towards providing confidence in the developed solution instead of actively looking for issues	[32]

5 Discussion

When aggregating the findings from the previously described papers, several insights can be derived that help to understand the issues that arise when trying to introduce TDD. This, in turn, also helps to identify potential measures on how to overcome these deficiencies.

One major challenge that became apparent was a lack of general knowledge, experience, and competencies regarding the domain of testing by many developers that were supposed to implement the projects in a test-driven manner, hindering them from creating efficient and effective automated tests [27], [30], [34], [35]. This coincides with the findings of a study [37] regarding the situation of teaching in software engineering. There, the authors compared the needs of the industry with the skills taught by universities to their students. They discovered that, while testing is one of the most relevant skills, big knowledge gaps still exist. Since the students themselves are oftentimes also not particularly interested in testing, conveying the necessary skills is a challenging task that might necessitate lecturers to come up with new and engaging approaches, such as the use of gamification [38].

Even though TDD provides excellent benefits, its application in the industry is a demanding task [30], with one of the main issues being the lack of previous exposure to TDD [30], [33].

Naturally, this is a significant factor because not only general testing skills are necessary, but also capabilities that are specific to TDD.

Heavily related to the aforementioned aspects is the challenge of switching to a TDD mindset for those that have extensive experience with the traditional TLD method. This not only requires high self-discipline but can also be accompanied by a huge (perceived) overhead [27], [28], [29], [31]. This impression can be exacerbated by the high volume of test code and the need to continuously maintain the test base [32].

These issues, in turn, force the developers to spend a lot of time familiarizing themselves with the corresponding processes and procedures, while simultaneously posing a risk of the method being incorrectly applied (due to a lack of skills or willingness) and therefore negatively affecting the results. In addition, one study [36] even concluded that previous exposure to unit testing negatively affects the perception of TDD. Therefore, the development of methods for the introduction of TDD to experienced testers as well as the exploration of further ways to make TDD more accessible appear to be extremely important to facilitate its (correct) application. The latter could, for instance, include tools for its teaching, process models, collections of case studies and best practices, experience reports, or workshop concepts. Moreover, in addition to the developers, senior management also must be convinced and educated on the method [27], [32]. Otherwise, the comparatively high time investment for creating test cases might be perceived as wasted and unproductive, leading to a mandated return to the traditional TLD method.

Furthermore, several researchers have found that the absence of a detailed design of the system is challenging when applying TDD [29], [31], [35]. This is another issue that could be at least somewhat alleviated through process models, guidelines, checklists, and best practices. While the specifics of different projects may obviously vary, preventing the creation of a universal design or similar approaches, providing the developers with general guidance that they can rely on to structure their own work appears to be a sensible solution to tackle the issue. This way, the negligence of important aspects can be reduced or even entirely avoided, since the work is conducted in a more structured manner. While not completely related, this focus on a better structured approach also pertains to the correct choice of implemented test cases, since they are oftentimes rather geared towards providing confidence in the developed solution instead of actively looking for issues [32].

Another aspect that became visible is a frequently prevailing lack of proper tools for (automated) testing. However, this can be probably seen as a combination of an actual absence as well as an insufficient knowledge of many developers regarding already existing tools that could be suitable for their respective tasks. Therefore, a comprehensive overview of such tools as well as an extensive body of published case studies might help to find inspiration on how to approach projects from a tooling perspective.

Finally, the existence of complex application cases that are not yet sufficiently studied in the context of TDD as well as the question how to deal with legacy code are additional challenges that call for further research and the exploration of innovative approaches.

6 Conclusion

With software having an increasingly important role in today's society and influencing nearly all parts of daily life in at least some capacity, it also becomes significantly more important to ensure its intentional functioning and the prevention of having undetected errors. Consequently, one of the major concerns of the software engineering community is the aspect of quality assurance. For this purpose, a plethora of tools, concepts, and approaches have been proposed. One of these is the concept of TDD, which changes the traditional order in which productive code and the corresponding tests are written. Thus, TDD also heavily influences the general software design of the developed solutions. In doing so, a more modular structure is created that also features a better test coverage, leading to improved product quality. However, this also comes with several challenges that can have a negative impact on the obtained results. Hence, it is important to be

aware of the potential issues which allows to account for these problems in practical projects and also provides guidance for researchers that strive to improve upon the current situation. For this reason, a structured literature review was conducted, focusing on the major challenges that occur when applying test-driven development in software engineering, aside from the already well-researched increase in development time and required effort. In doing so, several challenges that are related to the people participating in the development process, the related software and tooling, and the development process itself were identified. Furthermore, potential ways of overcoming those issues were discussed.

While many aspects are significant, some appear to offer particularly high potential for further studies. Especially improving the ways of teaching testing in general as well as TDD in specific, creating new tools suitable for the purpose, the exploration of ways on how to deal with legacy code when trying to utilize TDD, and the provisioning of process models, guidelines, and best practices that help the developers in correctly utilizing the TDD methodology to benefit their projects appear to be promising avenues for future research. As with any scientific publication, there are some limitations and potential weaknesses. For this article, these mostly pertain to the possibility that there is additional insightful literature that was not covered by the search process. Moreover, despite the best efforts to ensure objectivity and diligence, the human factor during the filter and analysis process might also contribute to a certain bias or oversights.

References

[1] R. Kazman and L. Pasquale, "Software Engineering in Society," *IEEE Softw.*, vol. 37, no. 1, pp. 7–9, 2020. Available: https://doi.org/10.1109/MS.2019.2949322

[2] W. E. Wong, N. Mittas, E. M. Arvanitou, and Y. Li, "A bibliometric assessment of software engineering themes, scholars and institutions (2013–2020)," *Journal of Systems and Software*, vol. 180, no. 11, p. 111029, 2021. Available: https://doi.org/10.1016/j.jss.2021.111029

[3] K. Beck, *Test-Driven Development: By Example*, 20th ed. Boston: Addison-Wesley, 2015.

[4] D. Staegemann, M. Volk, E. Lautenschlager, M. Pohl, M. Abdallah, and K. Turowski, "Applying Test Driven Development in the Big Data Domain – Lessons from the Literature," in *Proceedings of the 2021 International Conference on Information Technology (ICIT)*, Amman, Jordan, pp. 511–516, 2021. Available: https://doi.org/10.1109/ICIT52682.2021.9491728

[5] W. Bissi, A. G. Serra Seca 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, no. 4, pp. 45–54, 2016. Available: https://doi.org/10.1016/j.infsof.2016.02.004.

[6] Y. Rafique and V. B. Misic, "The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis," *IIEEE Trans. Software Eng.*, vol. 39, no. 6, pp. 835–856, 2013. Available: https://doi.org/10.1109/TSE.2012.28.

[7] C. M. Keet and A. Ławrynowicz, "Test-Driven Development of Ontologies," in *Lecture Notes in Computer Science, The Semantic Web. Latest Advances and New Domains*, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto, and C. Lange, Eds., Cham: Springer International Publishing, pp. 642–657, 2016. Available: https://doi.org/10.1007/978-3-319-34129-3_39

[8] K. Davies, C. M. Keet, and A. Lawrynowicz, "More Effective Ontology Authoring with Test-Driven Development and the TDDonto2 Tool," *Int. J. Artif. Intell. Tools*, vol. 28, no. 7, 2019. Available: https://doi.org/10.1142/S0218213019500234

[9] T. Slaats, S. Debois, and T. Hildebrandt, "Open to Change: A Theory for Iterative Test-Driven Modelling," in *Lecture Notes in Computer Science, Business Process Management*, M. Weske, M. Montali, I. Weber, and J. Vom Brocke, Eds., Cham: Springer International Publishing, pp. 31–47, 2018. Available: https://doi.org/10.1007/978-3-319-98648-7_3

[10] L. Crispin, "Driving Software Quality: How Test-Driven Development Impacts Software Quality," *IEEE Softw.*, vol. 23, no. 6, pp. 70–71, 2006. Available: https://doi.org/10.1109/MS.2006.157

[11] F. Shull, G. Melnik, B. Turhan, L. Layman, M. Diep, and H. Erdogmus, "What Do We Know about Test-Driven Development?" *IEEE Softw.*, vol. 27, no. 6, pp. 16–19, 2010. Available: https://doi.org/10.1109/MS.2010.152

[12] 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?” *IIEEE Trans. Software Eng.*, vol. 43, no. 7, pp. 597–614, 2017. Available: https://doi.org/10.1109/tse.2016.2616877

[13] L. Williams, E. M. Maximilien, and M. Vouk, “Test-driven development as a defect-reduction practice,” in *Proceedings of the 14th ISSRE*, Denver, Colorado, USA, 2003, pp. 34–45.

[14] D. Janzen and H. Saiedian, “Test-driven development concepts, taxonomy, and future direction,” *Computer*, vol. 38, no. 9, pp. 43–50, 2005. Available: https://doi.org/10.1109/MC.2005.314

[15] D. Janzen and H. Saiedian, “Does Test-Driven Development Really Improve Software Design Quality?,” *IEEE Softw.*, vol. 25, no. 2, pp. 77–84, 2008. Available: https://doi.org/10.1109/MS.2008.34

[16] R. S. Sangwan and P. A. Laplante, “Test-Driven Development in Large Projects,” *IT Prof.*, vol. 8, no. 5, pp. 25–29, 2006. Available: https://doi.org/10.1109/MITP.2006.122

[17] W. K. A. Law, “Learning Effective Test Driven Development – Software Development Projects in an Energy Company,” in *Proceedings of the First International Conference on Software and Data Technologies*, Setúbal, Portugal, pp. 159–164, 2006.

[18] M. Karlesky, G. Williams, W. Bereza, and M. Fletcher, “Mocking the Embedded World: Test-Driven Development, Continuous Integration, and Design Patterns,” in *Embedded Systems Conference*, San Jose, California, USA, 2007.

[19] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017. Available: https://doi.org/10.1109/ACCESS.2017.2685629

[20] Y. Levy and T. J. Ellis, “A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research,” *Informing Science: The International Journal of an Emerging Transdiscipline*, vol. 9, pp. 181–212, 2006. Available: https://doi.org/10.28945/479

[21] J. Webster and R. T. Watson, “Analyzing the Past to Prepare for the Future: Writing a Literature Review,” *MISQ*, vol. 26, no. 2, pp. xiii–xxiii, 2002.

[22] J. Vom Brocke, A. Simons, B. Niehaves, K. Reimer, R. Plattfaut, and A. Cleven, “Reconstructing the Giant: On the Importance of Rigour in Documenting the Literature Search Process,” in *Proceedings of the ECIS 2009*, Verona, Italy, 2009.

[23] J. Caba, F. Rincon, J. Barba, J. A. de La Torre, J. Dondo, and J. C. Lopez, “Towards Test-Driven Development for FPGA-Based Modules Across Abstraction Levels,” *IEEE Access*, vol. 9, pp. 31581–31594, 2021. Available: https://doi.org/10.1109/ACCESS.2021.3059941

[24] M. Ghafari, T. Gross, D. Fucci, and M. Felderer, “Why Research on Test-Driven Development is Inconclusive?” in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Bari Italy, pp. 1–10, 2020. Available: https://doi.org/10.1145/3382494.3410687

[25] E. Lautenschläger, “The Perception of Test Driven Development in Computer Science – Outline for a Structured Literature Review,” in *Lecture Notes in Business Information Processing, Business Information Systems Workshops*, W. Abramowicz, S. Auer, and M. Stróżyna, Eds., Cham: Springer International Publishing, pp. 121–126, 2022. Available: https://doi.org/10.1007/978-3-031-04216-4_13

[26] D. Staegemann, M. Volk, E. Lautenschläger, M. Pohl, M. Abdallah, and K. Turowski, “Applying Test Driven Development in the Big Data Domain – Lessons from the Literature,” in *Proceedings of the 2021 International Conference on Information Technology (ICIT)*, Amman, Jordan, pp. 511–516, 2021. Available: https://doi.org/10.1109/ICIT52682.2021.9491728

[27] J. Buchan, L. Li, and S. G. MacDonell, “Causal Factors, Benefits and Challenges of Test-Driven Development: Practitioner Perceptions,” in *Proceedings of the 2011 18th Asia-Pacific Software Engineering Conference*, Ho Chi Minh, Vietnam, pp. 405–413, 2011. Available: https://doi.org/10.1109/APSEC.2011.44

[28] A. Marchenko, P. Abrahamsson, and T. Ihme, “Long-Term Effects of Test-Driven Development A Case Study,” in *Lecture Notes in Business Information Processing, Agile Processes in Software Engineering and Extreme Programming*, P. Abrahamsson, M. Marchesi, and F. Maurer, Eds., Springer, Berlin, Heidelberg, pp. 13–22, 2009. Available: https://doi.org/10.1007/978-3-642-01853-4_4

[29] S. Hammond and D. Umphress, “Test driven development: the state of the practice,” in *Proceedings of the 50th Annual Southeast Regional Conference on - ACM-SE '12*, Tuscaloosa, Alabama, p. 158, 2012. Available: https://doi.org/10.1145/2184512.2184550

[30] A. Causevic, D. Sundmark, and S. Punnekkat, “Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review,” in *Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, Berlin, Germany, pp. 337–346, 2011. Available: https://doi.org/10.1109/ICST.2011.19

[31] S. Kollanus, “Critical Issues on Test-Driven Development,” in *Lecture Notes in Computer Science, Product-Focused Software Process Improvement*, D. Caivano, M. Oivo, M. T. Baldassarre, and G. Visaggio, Eds., Springer, Berlin, Heidelberg, pp. 322–336, 2011. Available: https://doi.org/10.1007/978-3-642-21843-9_25

[32] A. Causevic, S. Punnekkat, and D. Sundmark, “TDDHQ: Achieving Higher Quality Testing in Test Driven Development,” in *Proceedings of the 39th Euromicro Conference on Software Engineering and Advanced Applications*, Santander, Spain, pp. 33–36, 2013. Available: https://doi.org/10.1109/SEAA.2013.47

[33] R. Latorre, “A successful application of a Test-Driven Development strategy in the industrial environment,” *Empir Software Eng*, vol. 19, no. 3, pp. 753–773, 2014. Available: https://doi.org/10.1007/s10664-013-9281-9

[34] A. Nanthaamornphong and J. C. Carver, “Test-Driven Development in scientific software: a survey,” *Software Qual J*, vol. 25, no. 2, pp. 343–372, 2017. Available: https://doi.org/10.1007/s11219-015-9292-4

[35] I. Karac and B. Turhan, “What Do We (Really) Know about Test-Driven Development?” *IEEE Softw.*, vol. 35, no. 4, pp. 81–85, 2018. Available: https://doi.org/10.1109/MS.2018.2801554

[36] M. T. Baldassarre, D. Caivano, D. Fucci, S. Romano, and G. Scanniello, “Affective reactions and test-driven development: Results from three experiments and a survey,” *Journal of Systems and Software*, vol. 185, 2022. Available: https://doi.org/10.1016/j.jss.2021.111154

[37] V. Garousi, G. Giray, E. Tüzün, C. Catal, and M. Felderer, “Aligning software engineering education with industrial needs: A meta-analysis,” *Journal of Systems and Software*, vol. 156, pp. 65–83, 2019. Available: https://doi.org/10.1016/j.jss.2019.06.044

[38] G. Fraser, A. Gambi, M. Kreis, and J. M. Rojas, “Gamifying a Software Testing Course with Code Defenders,” in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Minneapolis MN USA, pp. 571–577, 2019. Available: https://doi.org/10.1145/3287324.3287471