*VAWKUM Transactions on Computer Sciences*
*http://vfast.org/journals/index.php/VTCS@ 2023 ISSN(e): 2308-8168, ISSN(p): 2411-6335*
*Volume 11, Number 1, January-June 2023* *pp: 268-280*

# Test Driven Development and Its Impact on Program Design and Software Quality: A Systematic Literature Review

**Dua Agha, Rashida Sohail, Areej Fatemah Meghji*, Ramsha Qaboolio, Sania Bhatti**

Department of Software Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan

***Keywords***: *Test-Driven Development, Literature Review, Code Coverage, Software Quality, Agile, XP.*

***Journal Info***:
*Submitted:*
*April 10, 2023*
*Accepted:*
*June 10, 2023*
*Published:*
*June 24, 2023*

**Abstract** Test-Driven Development (TDD) is a methodology in software development that necessitates tests to be written before to the production code. This approach can be used in any software development paradigm that involves writing code, including Agile, Scrum, XP, and Lean. This research paper surveys the impact of TDD on software development with a specific focus on its effects on code coverage, productivity, internal and external software quality, and the affective reactions associated with TDD. The paper also identifies potential challenges and drawbacks of implementing TDD, such as increased overhead and time consumption, a learning curve for developers, and difficulty in testing certain types of code. The studies' results suggest that TDD can improve code coverage, and code quality, reduce defects, increase productivity and developer satisfaction, improve internal and external software quality, and ultimately lead to higher customer satisfaction.

***Correspondence Author Email Address:**
areej.fatemah@faculty.muet.edu.pk

## 1 Introduction

Test-Driven Development (TDD) is an iterative software development methodology that emphasizes writing automated tests before writing actual production code. In contrast to traditional software development where software is produced first and test cases are created later, this approach interlaces programming, the creation of unit tests, and refactoring on the source code. In TDD, developers write a failing test case first, then write the code that passes the test intending to create a suite of tests that ensure the code behaves correctly [1]. When compared to traditional methods, TDD can deliver higher-quality projects in lesser time. TDD implementation needs developers and testers to precisely predict how the program and its features will be utilized in practice. An advantage of TDD is the generation of a regression-test suite as

a byproduct; this reduces human manual testing while detecting errors earlier, resulting in faster repairs. Also, as tests are performed at the start of the design cycle in this approach, the time and money spent on debugging at later stages is reduced. This is also why TDD is referred to as test-first development. It needs to be noted that refactoring plays a key role in the TDD process since it allows the improvement in the internal structure of the code and design whilst preserving the external behavior of the code [2]. TDD is also famously known by the red-green-refactor cycle.

The concept of TDD can be traced back to the early days of Extreme Programming (XP), an Agile software development methodology. Kent Beck, one of the founders of XP, is credited with popularizing TDD in his book "Test-Driven Development by Example" [1]. TDD gained wider acceptance in the software development community in the early 2000s and has since become an essential practice for many development teams [3] with reduced defects, increased productivity, and improvement in the external quality of the software. According to the State of Agile report (2018), 33% of teams follow this approach in their work [4]. TDD is compatible with various software development paradigms, including Agile, Scrum, XP, and Lean [5].

Implementing TDD in big data projects can be challenging due to the large volumes of data, complex systems, and the lack of suitable tools. A systematic literature review of the existing literature has been presented in this paper, adhering to the principles outlined by Kitchenham et al. [6]. A systematic review is an investigation in which a research issue, question, or hypothesis is addressed to collect evidence from various primary studies using a systematic analysis and data extraction procedure. The review investigation procedure began with the formulation of the research protocol, which defined the goal of the review. Nine research questions have been formulated to guide the research process. This paper aims to contribute to the existing literature on TDD by investigating its impact on program design and software quality. We also explore the impact of TDD on software development, focusing on its effects on code coverage, productivity, internal and external software quality, and affective reactions associated with TDD. The potential advantages, challenges, and drawbacks of implementing TDD are also discussed. Six knowledge bases comprising IEEE Xplore, ACM digital library, Springer Link, ScienceDirect, Google Scholar, and the Bulletin of Electrical Engineering and Informatics (BEEI) have been explored. A total of 261 papers were initially collected using the selected search strings. After following the literature review criteria, an analysis of 22 papers has been presented in this paper.

The rest of this paper is organized as follows: Section 2 presents the methodology of the systematic literature review followed by the discovered results presented in section 3. A discussion has been presented in section 4 followed by the conclusion of the paper.

# 2 Methodology

A systematic review is a research approach that aims to address a research question or hypothesis by systematically examining and extracting data from a range of primary studies [7]. A comprehensive review of the existing literature was carried out in this paper adhering to the principles outlined by Kitchenham et al. [6]. The review methodology has been depicted in Figure 1.

For conducting the systematic literature review following procedure was adopted.

1. During the plan phase of a systematic review, it is essential to define the research questions that the review will address and create a review protocol that outlines the basic review procedures. The review protocol should include the sources used to identify primary studies, the criteria for inclusion and exclusion, and how these criteria will be applied.
2. In the conduct phase of the review, data is extracted according to the research questions, and the collected data is synthesized and evaluated.
3. The final phase of the systematic review involves reporting the results of the review. This phase involves writing up the analysis, which will ensure the credibility and usefulness of the systematic review.

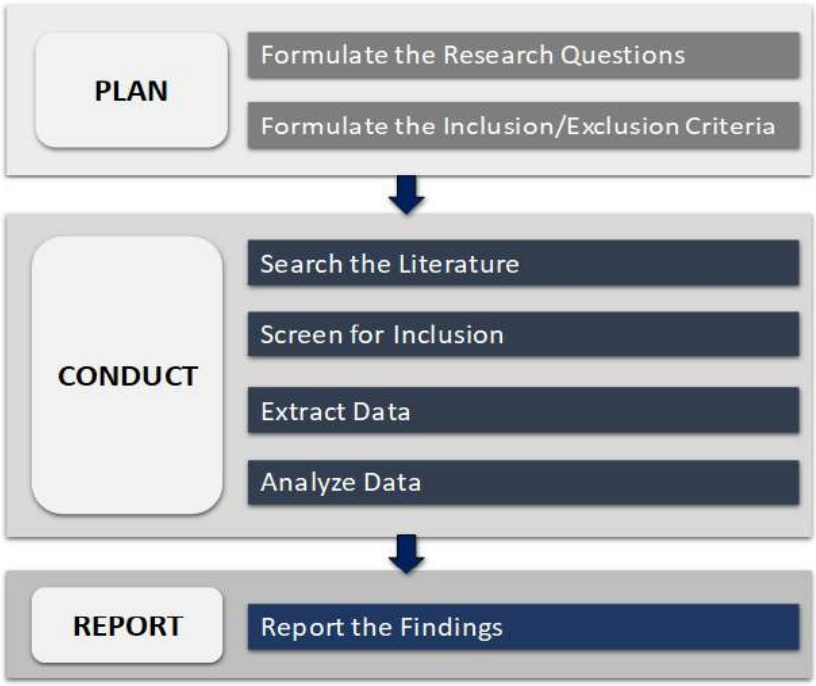


**Figure 1.** Review Methodology

## 2.1 Phase-1 Plan

This paper is designed to answer the following nine research questions. These questions assist in determining how TDD aspects impact Software Practices.

RQ1: What is the impact of TDD on code coverage?
RQ2: In which software development paradigms can TDD be utilized?
RQ3: What are the affective reactions associated with TDD, and how do they affect the software development process?
RQ4: What is the impact of TDD on software development productivity, as well as internal and external software quality?
RQ5: What are the primary benefits of implementing TDD in software development?
RQ6: What are some of the potential challenges of implementing TDD in software development?
RQ7: What are some of the potential drawbacks of implementing TDD in software development?
RQ8: How can TDD be integrated into existing development processes and methodologies?
RQ9: Why is the application of TDD in the domain of Big Data engineering considered a new proposition?

The search for the answers pertaining to the raised research questions was carried out in the six knowledge bases namely IEEE, ACM, Springer, Science Direct, Google Scholar, and BEEI. The search was carried out using the search strings outlined in Table 1.

**Table 1.** Search Strings Performed on Different Knowledge Bases

| Knowledge Base | Search Strings |
|---|---|
| IEEE | "Test Driven Development", "Test Driven Development" AND "Software Quality", "TDD" AND "Program design", "Test Driven Development" AND "Challenges" |
| ACM | "Test Driven Development", "Test Driven Development" AND "Software Quality", "TDD" AND "Program design", "Test Driven Development" AND "Code", "Test Driven Development" AND "Software Development" |
| SPRINGER | "Test Driven Development", "TDD" AND "Program design", "Test Driven Development" AND "Software Quality", "TDD" AND " Software Development ", "Test Driven Development" AND "Effects", "Test Driven Development" AND "Challenges" |
| SCIENCE DIRECT | "Test Driven Development", " TDD" AND "Program design", "Test Driven Development" AND "Software Quality", "Test Driven Development" AND "Code" |
| BEEI | "Test Driven Development" AND "Software Development" |
| GOOGLE SCHOLAR | "Test Driven Development", "Test Driven Development" AND "Software Quality", "Test Driven Development" AND "Software Development", "Test Driven Development" AND "Code", "Test Driven Development" AND "Challenges", "Test Driven Development" AND "Effects" |

After the formulation of the research questions and the selection of the knowledge bases, the next step was the design of the inclusion and exclusion criteria for the study. The inclusion and exclusion criteria for the review conducted in this paper have been outlined in Table 2.

**Table 2.** Inclusion and Exclusion Criteria for Paper Selection

| SNO | Inclusion Criteria | Exclusion Criteria |
|---|---|---|
| 1 | Papers must be fully accessible | Papers that are not accessible |
| 2 | Referenced studies must provide the results based on TDD practice in Software Development and its impact on software quality | Papers that do not meet the research objectives |
| 3 | Papers that provide answers to the research questions | Papers that do not provide answers to the research questions |
| 4 | Papers must be published in reported journals or conference proceedings | Papers published in non-peer-reviewed sources (such as blogs, forums, or non-scientific publications) |

## 2.2 Phase-2 Conduct

The primary studies identified throughout the search process using the established technique were chosen following the suggested inclusion and exclusion criteria outlined in Table 2. Reviewing the content of the potential references was essential to determine their applicability to the current study and, primarily, if these studies related to TDD practices. The majority of the articles did not address any of the issues covered in this paper and were more relevant to other fields, thus had to be excluded. Initially, 261 papers were collected as shown in Table 3 using the search strings outlined in Table 1. In the next phase, 198 papers were removed after reading their titles. Following the inclusion and exclusion criteria mentioned in Table 2, 39 papers were analyzed, of which 22 were selected for systematic literature review, and 17 were excluded. The selected papers were then analyzed in detail to extract the answers to the research questions. Figure 2 shows the details of the paper selection process. Whereas Figure 3 shows a comparison chart of knowledge base coverage based on the chosen search strings.

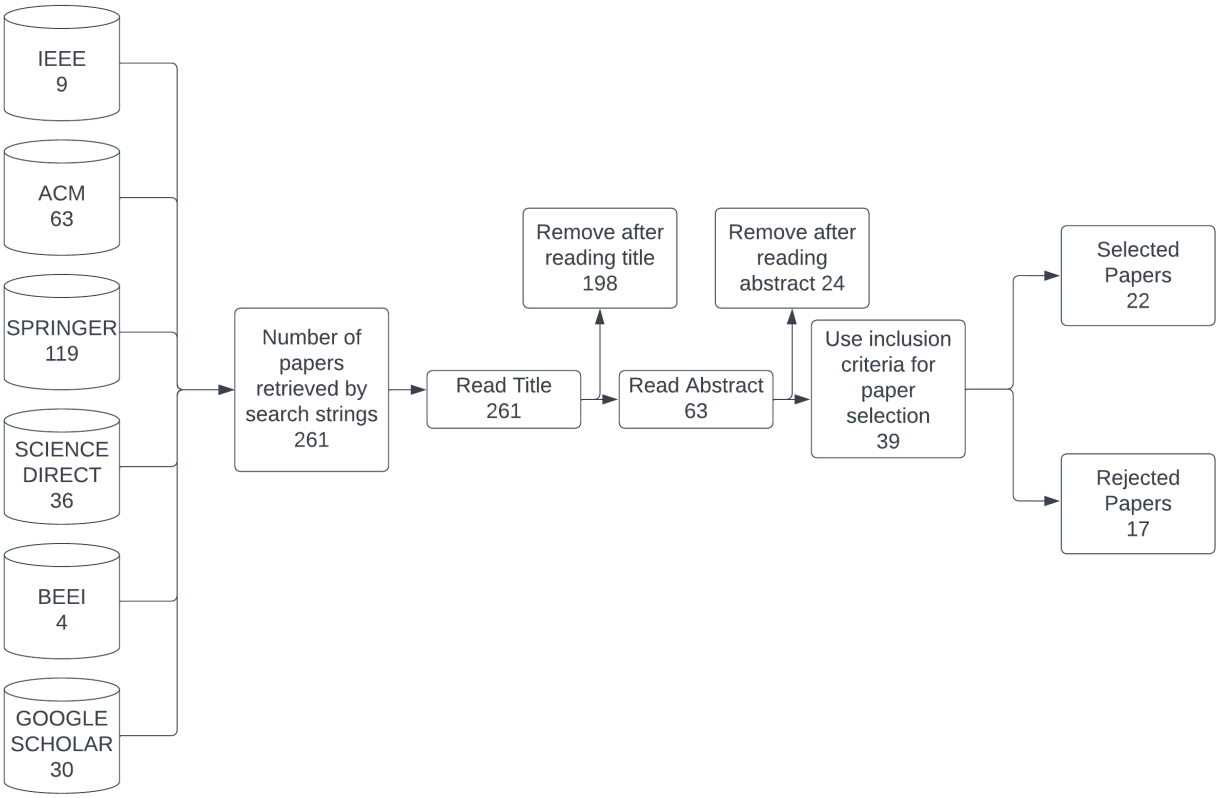


**Figure 2.** Paper Selection Process

**Table 3.** Number of Papers Collected from the Knowledge Bases

| Knowledge Base | No. of Research Papers |
|---|---|
| IEEE | 9 |
| ACM | 63 |
| SPRINGER | 119 |
| SCIENCE DIRECT | 36 |
| BEEI | 4 |
| GOOGLE SCHOLAR | 30 |
| TOTAL | 261 |

Figure 4 demonstrates that the number of articles on TDD has steadily increased over the past few years. This growth may be a result of the emphasis that academics and industry experts have placed on quality and the implementation of TDD inside software companies. Additionally, as shown in Figure 3, the first relevant research on TDD-related features that are addressed in this work started to appear in 2005.

## 2.3 Phase-3 Report

The nine research questions in the light of a systematic literature review are analyzed in this phase. TDD effectiveness, potential benefits, drawbacks, challenges, and TDD integration into existing development

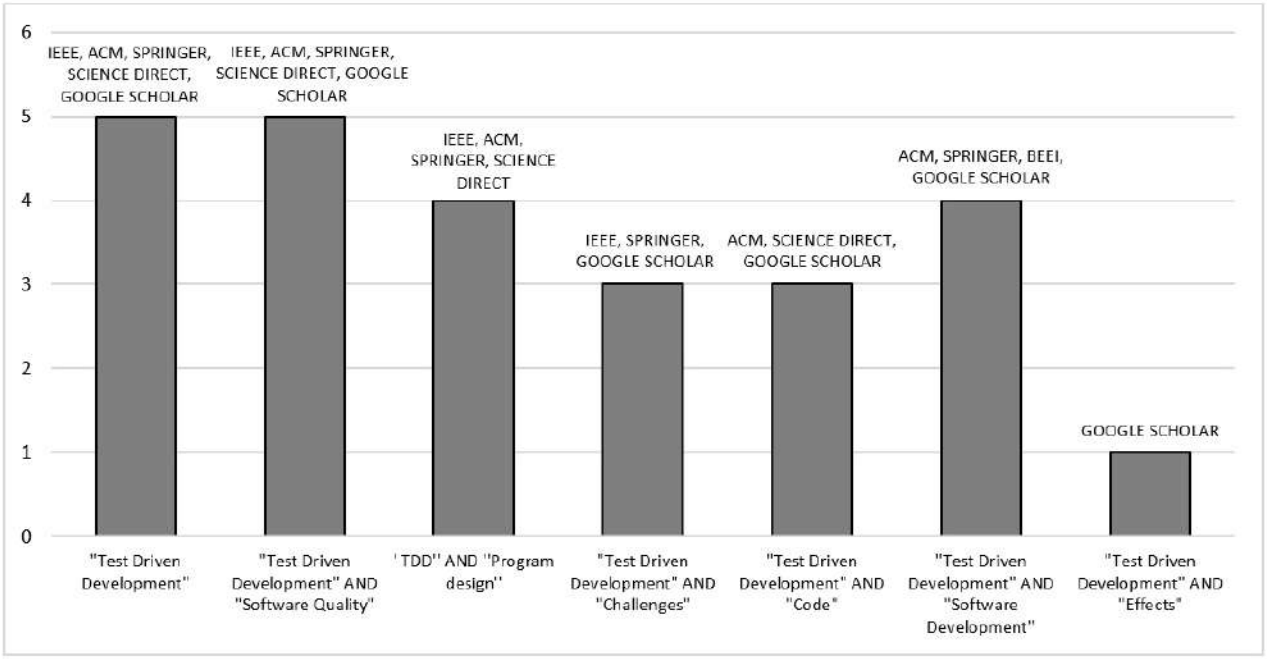


**Figure 3.** Database Coverage of Search Strings

processes are also revealed in this phase.

# 3 Result Analysis

This section reports the answers to the research questions targeted in this review. A summary of the primary studies related to TDD in software development has been presented in Table 4.

## 3.1 Impact of Test Driven Development Code Coverage – RQ1

Taufiqurrahman et al. examine the impact of TDD on code coverage [8]. The study found that TDD improves code coverage because all methods and functionalities are tested, and the higher the code coverage, the higher the software quality will be. According to Santos et al., the use of TDD has a positive impact on the productivity of developers and results in increased quality of code and tests [9]. The study also uncovered the use of TDD leading to a reduction in defects and improvement in code maintainability. TDD practices involve writing tests before implementing the source code, ensuring that the tests cover the desired functionality [10]. This approach encourages thorough testing as developers continuously add tests and refactor the code during the implementation process. However, the studies emphasize the necessity of further research to explore the impact of TDD on different types of projects.

## 3.2 TDD Utilization in Software Development Paradigms – RQ2

TDD can be utilized in any software development paradigm that involves writing code. This includes both traditional as well as modern software development methodologies [11]. TDD can be used in different software development paradigms such as Agile, Scrum, XP, and Lean. TDD is a critical component in ensuring that software is developed and delivered to meet customer requirements and maintain high-quality standards in these paradigms [5]. According to Taufiqurrahman et al., the TDD practice is applied to various

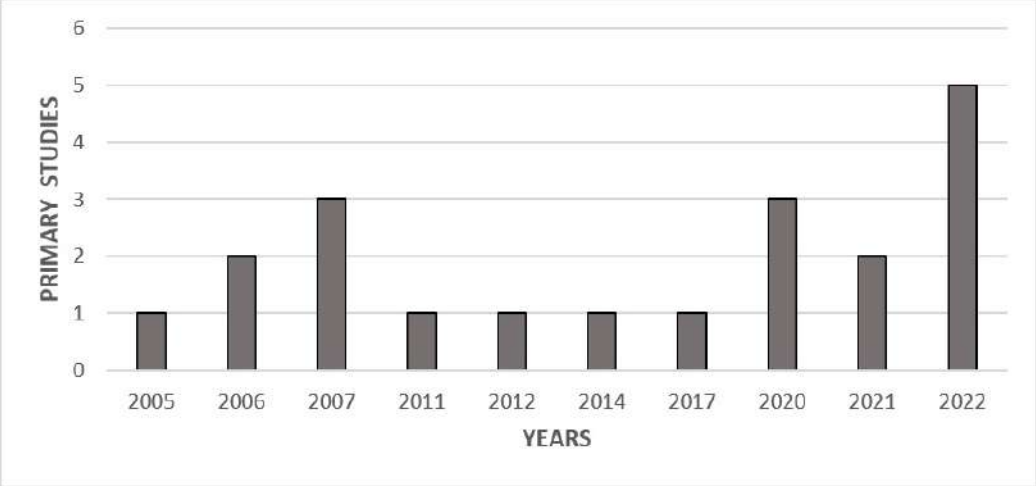


**Figure 4.** Temporal View of Publications

development paradigms, including agile development, extreme programming, and iterative development [8]. The TDD practice is compatible with these paradigms because it emphasizes the importance of testing and feedback loops during development. TDD can be utilized in any software development paradigm that involves writing code. This includes both traditional software development methodologies [11].

## 3.3 Affective Reactions Associated with TDD and Their Effect on the Software Development Process – RQ3

Nwandu et al. suggest that TDD positively affects software developers by increasing confidence in code quality, satisfaction, and productivity, and reducing stress [12]. TDD motivates developers to write tests for all code aspects, which improves code quality and reduces defects. Moreover, it reduces debugging and testing time and cost, accelerating product release and increasing customer satisfaction [11]. According to Tosun et al. the main effects achieved by applying TDD practice to software development are increased code coverage, improved external quality, and increased productivity [13]. TDD has increased developers' confidence by providing a safety net of tests, preventing code updates from breaking existing functionality. This has reduced stress by catching defects early and minimizing debugging efforts. TDD's collaborative nature, facilitated by shared language and artifacts, has improved team collaboration, leading to better code understanding and extensibility. Furthermore, TDD has motivated developers, resulting in higher job satisfaction, engagement, and a preference for this approach [14]. Thus, TDD can improve communication between developers, facilitate knowledge sharing, reduce anxiety, and increase confidence in the code. Although, it needs to be noted that factors such as team size and project complexity, can influence the effectiveness of TDD. The same can also be stated for the organizational context in which the approach is implemented.

## 3.4 Impact of TDD on Software Development Productivity and Internal and External Software Quality – RQ4

A review of the studies establishes that TDD has an overall positive impact on software development. This can be seen in an improvement in the quality of code, reduction in defects, and increased productivity as well as developer satisfaction [9]. TDD improves productivity by reducing the length of time spent

debugging and fixing bugs. It also improves internal software quality by promoting the development of modular, maintainable, and defect-free code [15]. Apart from the improvement in internal quality, external software quality is improved by catching defects early on in the development process. This directly leads to better user experience as well as an increase in customer satisfaction [11]. The review suggests TDD improves code coverage [8], reduces anxiety, and increases confidence in the production code [12], directly improving external quality [13]. Together, these studies suggest that TDD has a positive impact on software quality. Overall, this is an effective approach for software development, but careful planning and implementation are necessary for success [9, 16].

## 3.5 Primary Benefits of Implementing TDD – RQ5

TDD has several benefits in software development, including catching bugs early and ensuring that changes or modification to code does not add new bugs [11]. The implementation of TDD in software development has the potential to drive several benefits including but not limited to increased productivity, improved internal and external software quality, and better software design. TDD can prevent defects early in the development process, saving time and improving the user experience. Moreover, it encourages developers to write testable, maintainable, and loosely coupled code, leading to more modular and flexible software architectures.

## 3.6 Potential Challenges of Implementing TDD – RQ6

The reviewed literature identified three key aspects: the advantage of TDD in big data projects, challenges in implementing this approach in such projects, and the impact of TDD on project quality. Through the analysis presented in the papers, it can be concluded that TDD is beneficial for big data projects. The use of the approach ensured the maintenance of data quality, improved performance, and a reduction in defects. The main concern was how TDD would be incorporated with the lack of suitable tools in complex systems with the sheer volume of the data being considered [17]. Another concern while implementing TDD in software development comes with increased cost and time, handling complexity in large-scale systems, the potential for errors, a learning curve for developers, the need for effective collaboration, ensuring comprehensive test coverage, and maintaining tests over time. However, these challenges can be mitigated through proper planning, training, and experience. Despite the difficulties, TDD can enhance software quality and maintainability in the long run [18]. Encompass changes in established procedures, initial productivity impact, software design implications, test coverage and complexity, and adoption and cultural resistance. Adopting TDD requires adjustments to established procedures and mindsets, potentially leading to lower initial productivity due to the additional effort in writing tests upfront. TDD can influence software design decisions and necessitate a test-driven mindset. Ensuring comprehensive test coverage for various scenarios and dealing with complex cases pose challenges. Additionally, introducing TDD may encounter resistance from stakeholders and cultural barriers that need to be addressed to facilitate adoption [19].

## 3.7 Potential Drawbacks of Implementing TDD – RQ7

The potential drawbacks of this approach include the increase in overhead and added time, the learning curve for developers, difficulty in testing certain types of code, the possibility of a false sense of security, and the maintenance of tests [11]. Writing tests before writing code can be time-consuming and require developers to ensure that all necessary scenarios are covered. Developers may need to learn new skills and tools to effectively implement TDD, resulting in a temporary decrease in productivity.

## 3.8 Integration of TDD into Existing Development Processes and Methodologies – RQ8

TDD can be integrated into agile development processes by educating the team, starting with small and low-risk features or modules, creating test cases before coding, automating testing, refactoring code, and

fostering collaboration and communication between team members. By following these steps, teams can gradually incorporate TDD practices and benefit from improved code quality, reduced defects, and increased productivity [3] and TDD can also be integrated into existing development processes and methodologies by aligning it with agile methodologies like XP, applying it as a micro process within process models such as Scrum or DevOps, incrementally adopting TDD practices, promoting collaboration and communication among team members, leveraging TDD to support refactoring and code improvement, and utilizing TDD-friendly tools and frameworks. Integration requires a mindset shift, education, and gradual implementation [20].

## 3.9 Application of TDD in the Domain of Big Data Engineering Considered a New Proposition – RQ9

The application of TDD in big data engineering is considered a new proposition. While TDD has been extensively used in software development, its adaptation to the complexities of big data applications is recent. Big data applications have unique challenges due to the large volume, velocity, variety, and variability of data. Implementing and integrating big data requires sophisticated solutions beyond traditional techniques. The complexity of development and the need for accurate data analysis make TDD valuable in ensuring the quality and reliability of big data applications [18]. Big data applications pose challenges that traditional testing approaches may not adequately address, requiring specialized testing methodologies. TDD offers a promising solution with its iterative and incremental approach, focusing on writing tests first. By embracing TDD, organizations can enhance reliability, reduce errors, and increase confidence in the outputs of their big data applications [21].

Table 4 lists the 22 studies, which are divided into nine aspects based on the research questions. It is important to note that many studies contribute to multiple aspects. In those situations, the study was categorized using the factor that received the most attention in the paper.

**Table 4.** Primary Studies Related to Research Questions on TDD in Software Development

| RQ | Aspect | IEEE | ACM | SPRINGER | SCIENCE DIRECT | GOOGLE SCHOLAR | BEEI | TOTAL |
|---|---|---|---|---|---|---|---|---|
| RQ1 | Code Coverage | [8][9][10] | | | | | | 3 |
| RQ2 | Software Development Paradigms | [5][8] | | [11] | | | | 3 |
| RQ3 | Developers' Affective Reactions and Effect on the Software Development Process | [14] | | [11][13] | [12] | | | 4 |
| RQ4 | Impact on Software Development Productivity, Internal and External Quality | [8][9] | | [11][13][16] | [12] | | [15] | 7 |
| RQ5 | Benefits of Implementation | | | [11] | | | | 1 |
| RQ6 | Potential Challenges | [18] | | | | [19] | [17] | 3 |
| RQ7 | Potential Drawbacks | | | [11] | | | | 1 |
| RQ8 | TDD Integration into Existing Development Processes and Methodologies | [20] | | [3] | | | | 2 |
| RQ9 | Application of TDD in the Domain of Big Data Engineering | [18] | | | | [21] | | 2 |

# 4 Discussion

The effects of TDD have been the subject of numerous studies. The studies discussed in this paper, through comparative analysis, offer a deeper comprehension of TDD's effectiveness and potential drawbacks. They also provide valuable insights into its strengths and weaknesses. In addition, an investigation into how TDD practices influence testing efficiency and effectiveness by focusing on the quality of testing in TDD has been presented [22]. The review provides practitioners and researchers advice on how to improve testing quality by addressing obstacles and opportunities within the TDD approach [23].

TDD in software development offers advantages such as improved software quality through early defect detection and the creation of modular code [11]. It also boosts productivity by reducing debugging time. TDD positively affects developers, increasing confidence, job satisfaction, and productivity while reducing stress. It promotes thorough code testing, better communication, knowledge sharing, and overall codebase confidence [11]. The implementation of TDD also has positive results on a variety of performance metrics, such as development time, defect density, and customer satisfaction resulting in an overall improvement in project performance [20]. Also, although the norm has been for TDD to be implemented in agile software development, it can be utilized beyond that specific application domain. As discussed by Staegemann et al., TDD has the potential to be used in Big Data projects [21]. For instance, during the mock preparation stage, metrics of communication speed or confines of a maximum volume of the data can be explored to highlight unsuitable constructs at an earlier stage of the development process. This approach will also lead to high test coverage, conjoined with the opportunity for autonomous cooperation of distributed teams as well as increased flexibility [10].

The drawbacks of TDD include increased overhead and challenges in testing certain code types [11]. TDD's effectiveness depends on factors like team size, project complexity, and organizational context, requiring careful planning [17, 22]. Also, writing tests before writing the production code is time-consuming and requires an assurance that all necessary scenarios are covered. With newer technologies and platforms, developers will be required to learn new skills and tools to effectively implement TDD, resulting in a temporary decrease in productivity. The implementation of TDD can also be difficult in big data projects; with the volume of the data and the lack of effective tools making the implementation of the approach a challenge [21].

Although TDD can be utilized in various development practices, the impact may vary necessitating further research. An important point to consider is the use of the metric of code coverage. A 100% coverage indicates that the production code has been thoroughly tested, but this metric cannot ensure that the tests have been carried out correctly or that the system is free of defects [24]. As a result, additional measures need to be explored to see if internal software quality is improving. Also, factors such as technology and language used need to be explored [23].

The studies emphasize positive effects on developers, increased productivity, and improved software quality [25]. The studies have also examined TDD as an approach to decrease defect density. However, challenges such as increased overhead and testing difficulties exist. Further research is needed to understand the varying impacts of TDD, specifically on software design, and to explore additional measures for assessing software quality. Features such as extensibility, reusability, and maintainability should also be empirically researched. The implementation of TDD in different contexts, including big data projects, also requires careful consideration and adaptation.

# 5 Conclusion

TDD is a widely adopted software development approach that offers significant benefits to software development, including improved code coverage, productivity, internal and external software quality, and affective reactions associated with TDD. However, careful planning and implementation are necessary for success. Potential challenges and drawbacks of implementing TDD, such as increased overhead and time consumption, a learning curve for developers, and difficulty in testing certain types of code, must also be considered. Despite these challenges, TDD can positively impact software quality and customer satisfaction, making it an effective approach to software development. Future research on TDD in software

engineering could focus on exploring its impact on specific types of software development projects, such as large-scale enterprise projects or mobile application development. Additionally, more research could be conducted on the effectiveness of TDD in combination with other software development methodologies and practices. Moreover, there is a need for research on the impact of TDD on the long-term maintainability and sustainability of software projects.

# Author Contributions

**Dua Agha**: Conceptualization, Methodology, Writing-Original draft preparation. **Rashida Sohail**: Data curation, Visualization, Writing- Original draft preparation. **Areej Fatemah Meghji**: Visualization, Writing-Reviewing, and Editing. **Ramsha Qaboolio**: Data curation, Original draft preparation. **Sania Bhatti**: Reviewing, Supervision.

# Compliance with Ethical Standards

It is declared that the authors do not have any conflict of interest. It is also declared that this article does not contain any studies with human participants or animals performed by any of the authors. Furthermore, informed consent was obtained from all individual participants included in the study.

# Author Information

**ORCID:**
Areej Fatemah Meghji: 0000-0002-7302-2767

# References

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

[2] M. T. Baldassarre, D. Caivano, D. Fucci, N. Juristo, S. Romano, G. Scanniello, and B. Turhan, "Studying test-driven development and its retainment over a six-month time span," *Journal of Systems and Software*, vol. 176, p. 110937, 2021.

[3] 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.

[4] C. VersionOne, "2018 state of agile report," 2018.

[5] M. Siniaalto and P. Abrahamsson, "A comparative case study on the impact of test-driven development on program design and test coverage," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE, 2007, pp. 275–284.

[6] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

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

[8] F. Taufiqurrahman, S. Widowati, and M. J. Alibasa, "The impacts of test driven development on code coverage," in *2022 1st International Conference on Software Engineering and Information Technology (ICoSEIT)*. IEEE, 2022, pp. 46–50.

[9] A. Santos, S. Vegas, O. Dieste, F. Uyaguari, A. Tosun, D. Fucci, B. Turhan, G. Scanniello, S. Romano, I. Karac *et al.*, "A family of experiments on test-driven development," *Empirical Software Engineering*, vol. 26, pp. 1–53, 2021.

[10] L. Madeyski and Ł. Szała, "The impact of test-driven development on software development productivity—an empirical study," in *Software Process Improvement: 14th European Conference, EuroSPI 2007, Potsdam, Germany, September 26-28, 2007. Proceedings 14*. Springer, 2007, pp. 200–211.

[11] 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, p. 111154, 2022.

[12] I. C. Nwandu, J. N. Odii, E. C. Nwokorie, and S. A. Okolie, "Evaluation of software quality in test-driven development: A perspective of measurement and metrics," *Information Technology and Computer Science*, vol. 6, pp. 13–22, 2022.

[13] A. Tosun, O. Dieste, D. Fucci, S. Vegas, B. Turhan, H. Erdogmus, A. Santos, M. Oivo, K. Toro, J. Jarvinen *et al.*, "An industry experiment on the effects of test-driven development on external quality and productivity," *Empirical Software Engineering*, vol. 22, pp. 2763–2805, 2017.

[14] V. Bakhtiary, T. J. Gandomani, and A. Salajegheh, "The effectiveness of test-driven development approach on software projects: A multi-case study," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 5, pp. 2030–2037, 2020.

[15] A. Santos, S. Vegas, O. Dieste, F. Uyaguari, A. Tosun, D. Fucci, B. Turhan, G. Scanniello, S. Romano, I. Karac *et al.*, "A family of experiments on test-driven development," *Empirical Software Engineering*, vol. 26, pp. 1–53, 2021.

[16] 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.

[17] S. Mäkinen and J. Münch, "Effects of test-driven development: A comparative analysis of empirical studies," in *Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering: 6th International Conference, SWQD 2014, Vienna, Austria, January 14-16, 2014. Proceedings 6*. Springer, 2014, pp. 155–169.

[18] A. Causevic, S. Punnekkat, and D. Sundmark, "Quality of testing in test driven development," in *2012 Eighth International Conference on the Quality of Information and Communications Technology*. IEEE, 2012, pp. 266–271.

[19] M. U. B. Pervez, L. Eman, and B. D. Abbas, "Test driven development: A review."

[20] D. Staegemann, M. Volk, N. Jamous, and K. Turowski, "Exploring the applicability of test driven development in the big data domain," in *ACIS 2020 Proceedings*, 2020.

[21] 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 *2021 international conference on information technology (ICIT)*. IEEE, 2021, pp. 511–516.

[22] M. Siniaalto and P. Abrahamsson, "A comparative case study on the impact of test-driven development on program design and test coverage," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE, 2007, pp. 275–284.

[23] L. Crispin, "Driving software quality: How test-driven development impacts software quality," *IEEE software*, vol. 23, no. 6, pp. 70–71, 2006.

[24] D. Staegemann, M. Volk, M. Perera, C. Haertel, M. Pohl, C. Daase, and K. Turowski, "A literature review on the challenges of applying test-driven development in software engineering," *Complex Systems Informatics and Modeling Quarterly*, no. 31, pp. 18–28, 2022.

[25] V. S. Bhadauria, R. K. Mahapatra, and S. P. Nerur, "Performance outcomes of test-driven development: an experimental investigation," *Journal of the Association for Information Systems*, vol. 21, no. 4, p. 2, 2020.