A COMPARATIVE STUDY ON EFFECT OF TEST-DRIVEN DEVELOPMENT ON SOFTWARE QUALITY

Wumi AJAYI[1], Olusola FATOYE[2], Taye OLAKU [3], Oluwatimilehin OLUWAGBEMI [4]
[1]Software Engineering Department, Babcock University, Ilisan Remo, Ogun State Nigeria,
[2-4] Computer Science and Information Science Department, Lead City University, Ibadan. Oyo State Nigeria,
[1] wumiajayi1@yahoo.com [2] olusolafatoy@gmail.com, [3]tayeolaku@gmail.com,
[4]timioluwagbemi@gmail.com

Keywords: Software Development, Software Quality, Test-Driven Development, Defects Detection, Code Coverage, Maintainability, Reliability, Team Composition, Developer Experience, Project Characteristics.

Abstract: *The development of software is an important aspect of modern technology, and to ensure that these technologies succeed, it is necessary to have good quality software produced. Test-driven development (TDD) is a popular approach that aims to improve software quality by writing tests before writing the actual code. A wide range of studies have been carried out on the effectiveness of TDD and there is a general lack of consensus regarding its impact on software quality. This review study used a systematic literature search to investigate the impact of Test-Driven Development (TDD) on software quality. Our results show that TDD and non-TDD approaches must be compared in developing software using a set of standardized measures.*

1. INTRODUCTION

Test-driven development (TDD) is a software development methodology involving automated tests before writing code. This process ensures that each piece of code undergoes a rigorous and validated test, which leads to higher-quality software with greater robustness, reliability, and stability. TDD has gained popularity over the years, and several software development firms have taken it as an established practice. According to [1], TDD is "a programming technique that uses automated tests to drive the development process" [1]. TDD is the writing of code that specifies its behavior and then runs tests to determine whether it meets these requirements. This constant process guarantees that code is tested at each stage of development, while all tests are completed for the code to be deemed complete.

Software quality refers to the features of software that influence its overall performance, dependability, usability, security, maintenance, and other desirable attributes. It is extremely important in software development. Good quality software is free of errors or defects, fulfills the requirements of users they are designed to serve, operates properly, and provides a positive user experience. To ensure that software applications function as they are designed, satisfy users' needs, and deliver value for stakeholders, the quality of software is essential. So, according to several authors, test-driven development (TDD) is one of the fundamental agile software development approaches that places a high priority on unit testing before writing production code [2]. [1] refers to TDD as one of the common Agile software development approaches which is derived from the agile principles and extreme programming (XP) of which XP consists of two test practices which are Test-Driven Development (TDD) and user acceptance testing (UAT) to ensure quality and alignment of developed software with the customer requirements.
The software development process is a complex one requiring careful planning and

implementation to ensure that the final product satisfies the desired quality standards. For determining the success and usability of software, appropriate design methods must be used to produce quality software.
One such method is Test-Driven Development (TDD), which involves writing tests before writing the actual code [3]. This approach aims to improve software quality by ensuring that the software meets the desired functionality and is free from errors. To assess the efficiency of TDD in enhancing software quality several studies have been carried out, but their results are inconclusive. Furthermore, the impact of TDDs on software quality still has not been agreed.
The objective of the present review is to give a comprehensive understanding of the comparative study on the impact of TDD on software quality to deal with these differences and weaknesses in the existing literature.
The paper proposes a study design and methodology to address some limitations of past studies, which will look at the current literature on TDDs and software quality, analyze methodologies used by prior researchers [4], and discuss how this research might have an impact.

Importance of software quality

In determining whether software applications work or fail, it is essential to assess their quality. There are some negative effects which include user frustration, decreased productivity, increased costs, and damage to the reputation due to poor software quality. Conversely, software that meets the highest standards of quality can have significant advantages in terms of enhancing user satisfaction, increasing productivity, and reducing costs as described [5].

Both researchers and practitioners have recognized the importance of software quality. The need to make software development processes more focused on Software Quality has been growing over the last few years [6]. One approach that has gained popularity is Test-Driven Development (TDD), which is aimed at improving software quality by ensuring that code is thoroughly tested and meets the desired functionality. Studies conducted have shown that there may be significant improvements in software quality as a result of TDD. Among others, it has been shown that TDD results in reduced defects and increased productivity against traditional development methods. In addition, a study conducted by [7] revealed that TDD has helped to improve software quality and shorten development time.

Despite the good results achieved by TDD, further research still needs to be done to fully understand its impact on software quality. The purpose of this review paper is to provide a full understanding of the comparative study concerning the impact of TDD on software quality.
This paper contributes to our knowledge of the efficiency of TDD as a method for improving software quality by bringing together current literature on TDI and Software Quality, which suggests a study design that addresses certain limitations identified in earlier studies.

The purpose/objective of the study

The purpose of this review paper is to conduct a comparative study on the effect of Test-Driven Development (TDD) on software quality. The review shall set out in detail the benefits and drawbacks of TDD, as well as give a summary of what researchers use to assess its efficiency for improving software quality.
A critical analysis of the current literature on TDD and software quality will be provided in this review paper to determine patterns, deficiencies, or inconsistencies within it.
The overall aim of this review paper is to assess the effectiveness of TDD in enhancing software quality through an objective, evidence-based assessment and define areas where further research and development should be carried out as a key area for software engineering.

2. LITERATURE REVIEW

Test-Driven Development (TDD) is a software development process that emphasizes the importance of testing throughout the development lifecycle. Increasing software quality by ensuring that the code meets the desired functionality and is not corrupted, involves programming tests before writing real code [1].
The TDD relies on an Agile Software Development Methodology, which focuses on

collaboration, flexibility, and responsiveness to change [8].

The TDD process usually involves three steps: writing a failed test, creating the code to pass it, and then rewording this code to improve its design and maintainability [1].
The process is ongoing, requiring developers to constantly write and perform testing so that the code meets the requirements of functionality and quality.

Test-Driven Development Cycle

Refactoring plays an important role within TDD because it allows the continuous improvement of software quality, while also ensuring that there is no deterioration in its outside behavior.

However, the TDD comprises six fundamental steps to achieve its goals which are as follows (Fig. 1):

1. Write a test case that describes the function completely. To make the test cases the developer must understand the features and requirements using user stories and use cases.
2. Run all the test cases and make sure that the new test case fails.
3. Write the code that passes the test case
4. Run the test cases
5. Refactor code – This is done to remove duplication of code.
6. Repeat the above-mentioned steps again and again [9].

The first step involves writing test cases to ensure clarity from the developer's perspective, the second step is when we execute the test cases to validate their correctness i.e. at this point, as functionality is still in development, the test must not pass.
However, the test is not correct and needs to be updated if it passes. Writing your code is the 3rd step.
However, to pass the test, it is necessary to write as little code as possible.
Then, to confirm that the intended functionality has been implemented, all tests need to be performed.

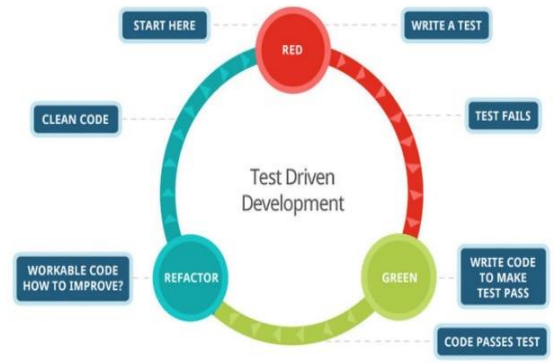

Fig. 1: Test-Driven Development [Source: https://www.linkedin.com/pulse/pros-cons-test-driven-development-tdd-taukir-hasan/]

Refactoring should improve the internal structure of the code as soon as all tests are completed.
In improving the quality of software, several studies have assessed the efficiency of TDDs with differing results. Despite mixed results, TDD is still a common approach for improving software quality in the context of an agile development environment. Several tools and frameworks aimed at facilitating the TDD process are in place to help the industry implement this approach [10]. Overall, TDD represents an important approach to improving software quality and is likely to continue to play a significant role in software development in the years to come.

Several studies have evaluated the effectiveness of Test-Driven Development (TDD) in improving software quality. A systematic review of 55 empirical studies on TDD and software quality, has shown that TDD can lead to improved software quality in terms of code coverage, defect density, and maintainability.

Nevertheless, some studies have shown that differences in software quality were not substantial across TDD and non-TDD approaches.
For instance, in a controlled study conducted by [9], the quality of software between TDDs and non-TDD systems was not significantly different. The impact of TDD on particular aspects of the quality of software has been evaluated in further studies.
For example, a study conducted by [10] shows that the adoption of TDD is leading to higher

customer satisfaction as well as faster delivery of features on an industrial project.

To evaluate the effectiveness of TDD in improving software quality, previous research studies on Test Driven Development and Software Quality have used various types of research methodologies. To compare the approaches of TDD and non-TDD, several trials have used controlled experiments. An example is that [10] carried out an uncontrolled experiment involving 24 development professionals and did not reveal any significant differences in the quality of software from TDD to non-TDD approaches. Similarly, there was a case study performed by [11] comparing software quality metrics from TDD Teams to the non-TDD teams on an Industrial Project.

The use of surveys and questionnaires has also been carried out for collecting data from software developers and interested parties in other studies. An empirical study was carried out in 20 software development teams to gather metrics on Software Quality as well as developer perceptions towards the Test Data Domain.

The mixed methods approach, including the combination of quantitative data collection and analysis methods, has also been applied in some studies. For example, to assess the effect of TDD on software quality in terms of code coverage, defect density, and maintainability, [9] applied both Quantitative and Qualitative data.

In general, the literature suggests that this type of tool can improve software quality and is also able to make a difference in terms of its effectiveness depending on factors such as the scope of the software development project, the developer's skills, and experience, or specific metrics used for measuring software quality.

The methods used in previous studies on software quality and TDD are different for each research question, target, or study context. In contrast to the common use of controlled trials and surveys as a tool for evaluating the effectiveness of TDD, other research methods which include case studies where combined methodologies are applied can give useful insights into the impact of TDDs on software quality.

2.1 BENEFITS OF TEST-DRIVEN DEVELOPMENT ON SOFTWARE QUALITY

Defect Intensity: In the development process, TDD has been shown to reduce defect intensity by detecting and repairing defects before they emerge; [7]. First, writing tests before they're embedded in the code base, helps identify problems and contributes to better software quality and lower error rates.

Programmer's Productivity: By providing more detailed guidance on what to do and serving as a reference document for future use, TDD can also increase the productivity of programmers. Studies have shown that reduced debugging time and better code clarity can lead to productivity gains through TDD. [10].

Time: While TDD may initially require additional time investment in writing tests, it can save time in the long run by reducing debugging and rework. TDD helps to identify and resolve problems early, thus minimizing the amount of time that defects are fixed during later phases of development [12].

Quality of code: By enforcing the conformity of defined requirements and promoting modular and maintainable code structures, TDD is intended to encourage better code quality. The TDD provides developers with guidelines for the production of focused and concise code that meets the required functionalities [13].

2.2 DRAWBACKS OF TEST-DRIVEN DEVELOPMENT ON SOFTWARE QUALITY

Refactoring/Learning Curve: The TDD may also introduce a learning curve of developers who have never been to this approach, potentially having an impact on productivity and code quality in the first place. It can take time to revise existing code to accommodate new tests, especially on complex projects as follows:

Maintainability: The lack of testing design or poor test coverage can result in maintainability problems, while the purpose of TDD is to enhance maintenance.

Cost: The upfront investment in writing tests and maintaining test suites can be perceived as an additional cost. However, studies have shown that the initial investment could be offset by the long-term benefits of improved software quality and reduced debugging efforts, [11].

Speed: Some studies suggest that, initially, due to the additional effort required to write tests, TDD may lead to a slight increase in development time. Eventually, though, TDD could lead to faster development by reducing the amount of debugging and reshuffling to increase its efficiency [7].

2.3 APPLICATIONS OF TEST-DRIVEN DEVELOPMENT

The relevance of using TDD in different fields and its impact are addressed in this section.

TDD in Embedded Software

Test-driven development is a reliable practice for embedded software engineering, as demonstrated by studies carried out on embedded systems [1]. In designing an integrated system, the tests are typically ad hoc and delayed up to the time when the hardware is developed. It is due to a lack of available required equipment during the design phase that can make it more difficult to test.

However, for the embedded systems it is essential to thoroughly test them because once they are put into production there will be a rapid increase in repair costs.

To make this possible, the TDD can therefore be facilitated by using mocking hardware techniques which include Interface Mock Reparations, Inheritance Mock Restraints, etc. According to them, programming software to an interface, rather than to a specific class itself, isolates it from the hardware. Virtual drivers are used to replace this.

TDD in Web Applications

According to the conclusions drawn by a web application developer using Spring Framework initially, TDD did not seem to be a good fit for bottom-to-top workflow. For example, if this sequence is followed: [1]

1. Creation of database tables and the domain model objects.
2. Repository layer implementation.
3. Service layer creation.
4. and finally the web layer

2.4 FINDINGS OF PREVIOUS STUDIES

Mixed results were obtained in prior studies on the development of software using a driven approach, as well as its impact on software quality.

TDD has been shown to improve software quality through the reduction of defects and increased code coverage in some studies. For example, in a controlled experiment conducted by [4], it was found that TDD teams produced code with 50% fewer defects than non-TDD teams.

However, a significant difference between TDD and non-TDD approaches in the quality of software has not been observed in other studies. Examples of this are a controlled experiment performed by [9], which did not reveal an overall difference in software quality among TDD and non-TDD teams. Similarly, a case study conducted by [10] finds no significant differences between TDD teams and non-TDD teams about software quality metrics.

It should be pointed out that various factors, such as the competence of the development team, the complexity of a project, and the testing frameworks and tools used may affect the efficiency of TDD in improving software quality. Overall, it has been shown that the relationship of TDD to software quality is complex and influenced by context.

The factors that influence the effectiveness of TDD for improving software quality need to be better understood through further research.

3. METHOD

The methodology used in this review paper incorporates a systematic literature review. To carry out this review, these measures will be adopted as follows:

1. Identifying relevant studies: Using relevant keywords such as "test-driven development", "software quality", "code coverage", "software defects", and "software testing", a comprehensive search will be carried out in academic databases such as Google Scholar, Research Gate, ACM Digital Library, and ScienceDirect. Furthermore, to identify further studies, reference lists of the corresponding literature will be drawn up.

2. Selection of studies: The inclusion criteria for selecting studies will be based on their relevance to the research question and the hypotheses. Only studies that directly investigate the relationship between TDD and software quality will be included. Studies that use other related methodologies such as Behavior-Driven Development (BDD) or Acceptance Test-Driven Development (ATDD) will be excluded.

A systematic approach to the review of literature will ensure that all relevant studies are identified and evaluated transparently.
This will also contribute to reducing the bias in the selection and analysis of studies, as well as gaining a comprehensive understanding of the relationship between TDDs and software quality.

3.1 Comparison of TDD and non-TDD approaches

Test-driven development (TDD) is an agile software development practice involving automated test cases before writing the code. Instead, approaches that do not use TDD are to first write a code and then create test cases so that it can be validated. To understand their impact on the quality of software, several studies have compared TDD and non-TDD approaches to software development.

A study performed by [11] compared TDD to traditional development non-TDD, demonstrating that it was associated with higher code quality, lower density of defects as well and better maintenance. Another study, conducted by [12], has shown that TDD is associated with an increased code quality and reduced density of defects in comparison to non-TDD.

Mixed or incomplete results have nevertheless been reported in some studies. However, in some cases, TDD did not lead to a significant reduction of the density of defects even when it was linked with enhanced code quality, as evidenced by an analysis carried out by [13].
A study [14], also found that, in some cases, TDD is linked to better code quality but does not give rise to any consistent improvement in its reliability. In general, although there is some evidence for the existence of a link between TDD and improved code quality and stability compared to other approaches based on non-TDD, results in this direction are not homogeneous, and further research is needed to understand the benefits and limitations of TDD.
This Review Paper implies that an effective approach to enhance the quality of software can be Test Driven Development (TDD), which in particular reduces the level of defects and improves code maintainability. The review emphasizes the need for careful consideration to be given to the possible advantages and disadvantages of TDD before its adoption as a software development practice.

3.2 Implications of the study

Moreover, the review identifies shortcomings in current literature which should be addressed as part of future research.
The effectiveness of TDD compared with other software development practices, e.g., should be examined in more rigorous studies. Research to explore the impact of TDD on software quality in a wide variety of contexts, e.g., as part of massive projects or organizations with different levels of experience has also been needed.

In summary, the findings of this review report will be valuable in understanding how TDD could contribute to software quality and highlight areas for future research.

3.3 Limitations of the Study

The review may be affected by publication bias, as studies with major results are likely to be published more frequently than studies without significant results.
The review did not examine other aspects related to software development, e.g., productivity or cost of projects, as it focused only on the impact of TDD in terms of Software Quality.

4. CONCLUSION AND RECOMMENDATIONS

The review paper examined the impact of Test-Driven Development (TDD) on software quality and compared it with non-TDD approaches.
The studies examined indicate that increased code quality and greater reliability as compared to non-TDD methods may be linked with the use of TDD. TDD includes automated testing before code writing that enables developers to detect defects before they occur during the development process, and makes sure codes are well designed and comply with specified requirements.

Studies showed that in comparison with non-TDD approaches, low defect density, higher code quality, and greater durability were linked to the use of TDD. However, some studies have shown mixed or weak results about the impact of TDD on software quality.

The results show that to improve the quality of software, TDD can be an effective approach but further research needs to be carried out to gain a full understanding of its benefits and limitations.
The efficiency of a TDD may be influenced by factors such as project size, team experience, and the particular context for software development. To this end, before using TDD as a software development practice, it is necessary to carefully consider the possible advantages and disadvantages.

Several recommendations for future research are made based on the limitations and shortcomings found in this review paper.
More empirical studies to investigate the effectiveness of TDD in comparison with various software development techniques, e.g., Agile Development or Traditional Software Development Processes, are needed for some reasons.

In addition, the impact of TDDs on other aspects of software development than its quality, such as productivity, project cost, and developer satisfaction, should be considered in subsequent research.

5. REFERENCES

[1]. Zeba Khanam & Mohammed Najeeb Ahsan (2017). Evaluating the Effectiveness of Test-Driven Development: Advantages and Pitfalls. International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7705-7716
[2]. Simone Romano et al. (2022). Do Static Analysis Tools Affect Software Quality when Using Test-driven Development? International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '22), September 19–23, 2022, Helsinki, Finland. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3544902.3546233.
[3]. Beck, K. (2003). Test-Driven Development: By Example. Addison-Wesley Professional.
[4]. Erdogmus H, Morisio M, Torchiano M (2005) On the effectiveness of the test-first approach to programming. IEEE Trans Softw Eng 31:226–237.
[5]. Al-Qutaish, R. (2013). Software quality: Definition, perspectives, and future directions. Journal of Software Engineering and Applications, 6(3), 43-52.
[6]. Fagan, M. (2013). Software quality engineering: A practitioner's approach. Springer Science & Business Media.
[7]. Janzen, D. S., & Saiedian, H. (2008). Test-driven development: Concepts, taxonomy, and future direction. Journal of Database Management (JDM), 19(4), 5-17.
[8]. Fowler, M., & Highsmith, J. (2001). The Agile Manifesto. Retrieved from https://agilemanifesto.org/.
[9]. Fucci, D., Turhan, B., Juristo, N., & Dieste, O. (2016). An external replication on the effects of test-driven development using a multi-site blind analysis approach. Empirical Software Engineering, 21(2), 607-644.
[10]. George, B., Williams, L.: A structured experiment of test-driven development. Information and Software Technology 46(5), 337–342 (2004) https://doi.org/10.1016/j.infsof.2003.09.011
[11]. George, B., Williams, L.: A structured experiment of test-driven development. Information and Software Technology 46(5), 337–342 (2004) https://doi.org/10.1016/j.infsof.2003.09.011
[12]. Nagappan, N., Maximilien, E. M., Bhat, T., & Williams, L. (2008). Realizing quality improvement through test driven development: Results and experiences of four industrial teams. Empirical Software Engineering, 13(3), 289-302.
[13]. Madeyski, L., & Kitchenham, B. (2015). Test-driven development: An empirical evaluation of agile practice. Journal of Systems and Software, 101, 235-253.

[14]. Ciolkowski, M., Jedrzejczyk, P., & Sliwerski, P. (2016). Does test-driven development improve software design quality? IEEE Transactions on Software Engineering, 42(8), 756-772.
.