ISSN No:-2456-2165

A Case Study of How Test-Driven Development Improves Software Quality

[1]Tharukshi Wickramasinghe
[1]Department of Information Technology
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

[2]Senal Walpola
[2]Department of Information Technology
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

[3]Tharindu Siriwardana
[3]Department of Information Technology Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

[4]Dilshan De Silva
[4]Department of Computer Science and Software Engineering Sri Lanka Institute of Information Technology
Malabe, Sri Lanka

Abstract:- The purpose of this study includes the Test-driven development (TDD) software development approach, which involves developing tests before implementing a feature into a software Application. With comparisons to Test Last Development (TLD), this study is focused on examining the effects of TDD on the productivity and quality of internal and external software. Results revealed an improvement in the quality of the internal software and a significant improvement in the quality of the external software. In general, TDD seems to result in the development of software that is more modular, simpler, and more tested. To properly comprehend how TDD affects software development, refer to the case study provided below.

***Keywoeds*:-** *Test-Driven Development (TDD), Software Quality, Software Testing, Software Development, Test.*

I. INTRODUCTION

Building, Releasing, and managing a more Reliable, usable, and maintainable software application is not easy. In order to build a well-performing application usually industry follows Software Development Life Cycle (SDLC). In that SDLC, testing the application which will match to the requirements of carry a major portion of the entire development to ensure software quality and manage future development. Software testing is crucial since it detects any issues or errors in the written code, allowing them to be corrected before the software product is released. Test-driven development (TDD) is a common agile approach in the software development business. Before source code is implemented, automated unit tests must be written gradually. Initially presented as an Extreme Programming (XP) component, it is currently utilized separately in the industry [1]. This makes the programmer consider the feature from various angles before writing any code for it. Additionally, it offers tests that programmers can run to make sure that newly added, updated, or refactored code doesn't disrupt any existing functionality. [2]

Even though Software Developers spend hours and hours building the application and don't spend much time focusing on testing the application will directly affect the software quality. Testing a properly defined test plan from the be- ginning of the project will bring advantages in a few main ways such as time to develop the properly working product with properly defined unit testing, black box testing, and white box testing will take less time compared to fixing bugs when the product development phase running and for the future use maintainability of the product also will be increased. As well as reliability, efficiency, and usability also can be ensured with a properly executed test plan.

Otherwise, developers would have to divide their time between creating new functions or projects and spending even more time fixing logical, functional, and other bugs. Because TDD basically begins with the development of test cases, the developer has a proper idea and knows before beginning to develop the actual function that these functions should pass these test cases in such a way and This is the actual requirement of the component, etc., adhering to the TDD approach is much simpler for the developer as well as the entire team. It would be much easier for the developer to maintain their concentration on the application development if they have a solid test plan before creating the application. in addition to assisting in eliminating self-doubts throughout the actual development process.

Finally, If the Development team can release a Properly tested and functionally verified system to customers, it will increase their trust and satisfaction. Therefore, the testing phase is compulsory to perform for any kind of software if that software needs to ensure that all the functionalities are working fine in the same way as the requirements.

➢ *History and Evolution of TDD*

The testing paradigms are the most obvious way that modern TDD varies from TDD in the early computing period. Modern TDD was streamlined through automated testing, although early TDD was typically an implementation of manual testing. Modern TDD was born using the Junit tool, even if Kent Beck's SUnit suite is widely regarded asthe pioneering TDD framework. The website JUnit.org was founded on August 16th, 2000, and shortly after that, on September 2nd, 2000, the NUnit framework was registered on SourceForge. On November 25th, 2000, JavaUnit was also registered on SourceForge. [3]

Java developers who preferred Agile or Extreme Programming as their preferred software development methodology were pleased with JUnit. Recently, a number of unit testing frameworks with architectures based on Kent Beck's SUnit have appeared. They are collectively referred to as the "xUnit class of tools," and they are available for almost all current programming languages, such as "CUnit" for C, "CppUnit"for C++, "RUnit" for R, etc.

➢ *TDD Implementation*

Understanding test-driven development's definition and practical use is crucial. TDD consists of four easy steps.

- Write a test that fails: This implies that before writing any code for a particular functionality, such as a method to calculate taxes, you should first write a test for that functionality, as well as the bare minimum of code that willbe needed to enable the test to actually run, such as a method definition. Given that this is how each test should start, it is the first and most crucial stage in TDD. [4]
- Write code to allow the test to pass: The developer now moves on to implementing the functionality needed for thetest to pass. The test is frequently run as the code is being written to determine which portion of the code is working. Most TDD practitioners believe that this real-time, immediate feedback actually increases developers' productivity. [4]
- Refactor the code (Optional): This is typically referred to as a TDD phase that is carried out after the test passes. Refactoring improves the clarity and precision of the code.[4]
- For whatever functionality that the programmer or developer wants to implement, they must go through the preceding processes again. [4]

➢ *Difference between Traditional Testing and TDD*

The test's first factor [5]. is the main distinction between TDD and conventional software testing. Since the entire code implementation is written out before any tests, the traditional software testing paradigm uses the test last approach, which views testing primarily as a verification mechanism to make sure that the implementation works as intended. The comparison between the typical testing cycle and TDD is shown inthe chart below. Here 1 shows how the modern TDD process in testing and 2 shows how the traditional testing process.

II. LITERATURE REVIEW

Test-driven development is a crucial aspect of the software development approach and agile methodologies. introduces how test cases created using the developer-first approach impact software quality. Currently, all the QA developers used some traditional methodologies such as traditional test-last approaches.

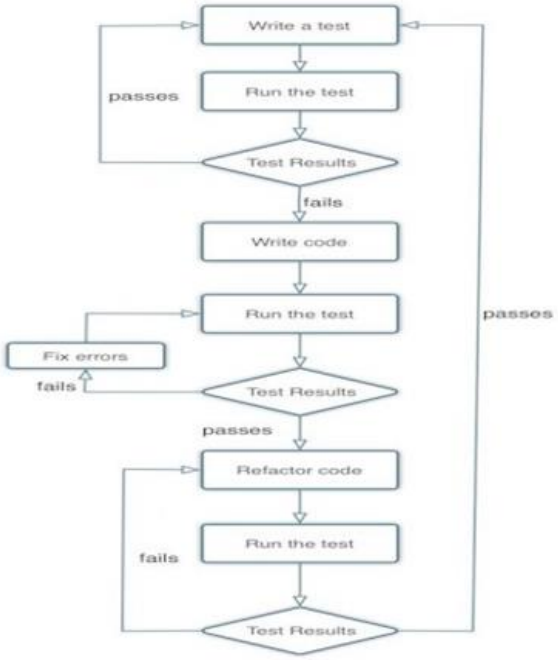

Fig 1 Modern TDD

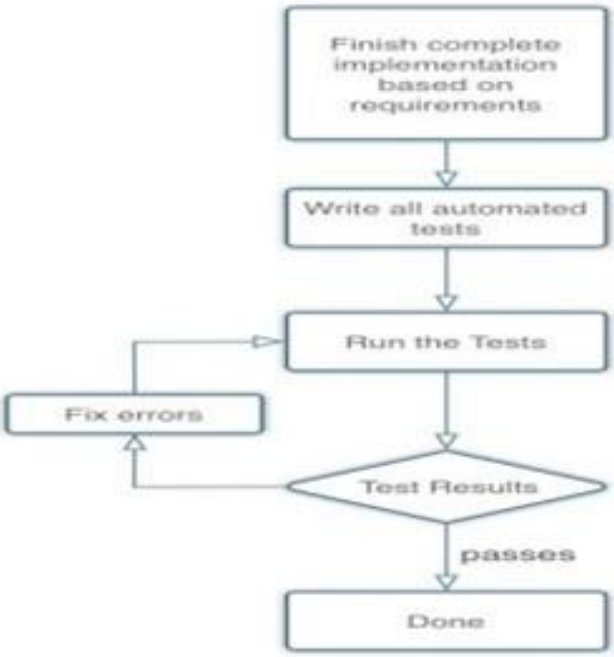

Fig 2 Traditional Testing

In traditional test-last approaches, all dev test all the test cases in the last of the software product application. In this research measures how this traditional approach and developer-first approach affect the software quality. In addition to that this research paper investigated the quality attributes for comparing the quality of the test-last and developer-first approaches. In this research, all the attributes are grouped as internal and external quality attributes for the test cases. Code coverage and Mutation coverage are used as internal quality attributes for the test cases. As external quality attributes, they are measuring the effectiveness of the code segment and the total number of defects found in source code. This research papers mainly focus on Agile methodology and how the quality improves when changing the traditional approach to developer-first approach. The result of this research is the experimental evaluation of data reveals that the test-first group's code is of higher quality than the test-last group.

[5] The ability of test-driven development (TDD) to raise the caliber of code has attracted a lot of interest from both practitioners and scholars. Unit testing is crucial to TDD even though it is largely a development practice. As a result, it can be considered a strong substitute for the Testing After Coding (TAC) method, which involves running unit tests after developing the code. A research team experimented at a Spanish software company to examine the quality and productivity differences between these two methodologies. The study discovered that while TDD slowed down the entire development process, it also improved the quality of unit testing. [5]

The ability of test-driven development (TDD) to raise the caliber of code has attracted a lot of interest from both practitioners and scholars. Unit testing is crucial to TDD despite the fact that it is largely a development practice. As a result, it can be considered a strong substitute for the Testing After Coding (TAC) method, which involves running unittests after developing the code. A research team experimented at a Spanish software company to examine the quality and productivity differences between these two methodologies. The study discovered that while TDD slowed down the entire development process, it also improved the quality of unit testing. [6] It should be highlighted, however, that TDD has advantages beyond unit testing since it can assist developers in identifying and resolving possible problems early in the development cycle, improving overall code quality and lowering maintenance costs. The two primary research questions are the primary focus of this study. analyzing the TDD and TAC (Test After Coding) productivity. Researchers carried out an experiment to assess the influence of Test-Driven Development (TDD) and Testing After Coding (TAC) methodologies on the accuracy and precision of unit tests. Prior to writing the actual code, TDD involves creating unit tests to ensure that it will work as intended. TAC, on the other hand, entails doing unit tests following the writing of the code. According to the study's findings, TDD, as opposed to TAC, produced a higher level of accuracy and precision in unit testing. This is due to the fact that TDD enables developers to find and fix problems early in the development process, resulting in more accurate and exact unit tests. TAC, on the other hand, may cause problems to go unnoticed and not be discovered until later stages, resulting in less accurate and precise unit tests. Overall, the study emphasizes the advantages of utilizing TDD versus TAC for enhancing the precision and accuracy of unit tests. [7]

Writing unit test cases before implementing the code is part of the Test-Driven Development (TDD) software development technique. 24 experienced pair programmers participated in a series of organized experiments to assess the effect of TDD on code quality. While the other group followed a waterfall-like process, one group used TDD to create a small Java program. Subject to questions about external validity, the experimental findings show that TDD programmers produced higher-quality code as demonstrated by the fact that they passed 18 percent more functional black-box test cases than the control group. The TDD programmers took 16 percent longer to do the work, though. The findings of additional statistical research showed a modest statistical link between the amount of time spent and the final code quality. Notably, the control group frequently skipped creating the necessary automated test cases after finishing their code, raising concerns about the effectiveness of waterfall-style approaches in encouraging sufficient testing. This finding supports the idea that TDD could lead to more unit testing being done in the software industry. [8]

The effectiveness of test-driven development (TDD), which is gaining popularity in software development, has been assessed in the context of web-based system development. The goal-question-metric design strategy was used to create the study, making it simple to replicate it in various industrial settings even with a small number of participants. TDD had a favorable effect on software development productivity, according to the study. In comparison to the test-last development approach, TDD was found to have a higher ratio of active development time defined as the time spent writing code to total development time. This implies that TDD motivates programmers to produce higher-quality code more quickly, increasing software development productivity. Overall, the results are in favor of using TDD while creating software, especially when creating web-based systems.

III. CASE STUDY DESIGN – METHODOLOGY

We used a survey questionnaire to gather the data for this case study. The questionnaire will ask participants about their experiences with TDD and non-TDD software development processes, how they view software quality with TDD and non-TDD, the time, cost, and effort associated with both TDD and non-TDD software development processes, the kinds of bugs or defects they have encountered with TDD and non- TDD, and how they feel about the role of automated testing in TDD. We can effectively collect data from a wide number of people with

various levels of software development experience by employing a questionnaire.

The purpose of this study is to look into how Test-Driven Development (TDD) affects the caliber of software. Finding Software engineers with knowledge of both TDD and non-TDD software development techniques across diverse businesses are a key component of the study. A survey questionnaire will be used to gather information about participants' experiences with TDD and non-TDD software development processes, as well as information about their opinions on the role of automated testing in TDD and their perceptions of software quality, time, cost, and effort.

By comparing the outcomes of the TDD and non-TDD software development processes, it will be possible to spotany changes in the software quality, time, cost, effort, or types of bugs or defects that are found. The data will be examined to spot patterns and trends. Using quantitative data to backthe results and participant quotations to offer a qualitative viewpoint, the findings will be summarized and conclusions concerning the effect of TDD on software quality will bereached.

Based on the study's findings, suggestions for software development teams on how they might use TDD to raise the quality of their work will be given. The study's limitations, including those related to sample size, participant choice, and study scope, will also be explored. Finally, recommendations for additional study will be made to build on the study's findings, such as looking into how TDD affects particularbusiness sectors or software programs.

IV. RESULT AND DISCUSSION

A software development methodology called Test-Driven Development (TDD) places a strong emphasis on buildingautomated tests prior to writing actual code. Due to its capacityto increase software quality by identifying errors early in the development cycle, providing a safety net for the code, andmaking sure that modifications to the code do not cause new errors, TDD has grown in popularity over time.

Investigating the effect of TDD on software quality wasthe goal of our case study. A set of software engineers who employed TDD in their development process were given a questionnaire. The questionnaire covered a range of TDD- related topics, including how it affects software quality.

Our study's findings suggest that TDD improves the qualityof software. Ninety percent of the developers said TDD hasraised the caliber of their program. This result is in line withother research that demonstrated how TDD can result in fewer faults in the finished product and higher software quality. [9]

TDD can increase software quality in part because it makes it easier to find and correct defects earlier in the development cycle. Developers can find and correct faults before they become significant problems by writing tests first. As a result, the finished product has fewer flaws and is simpler to maintainand update over time.

Additionally, our research revealed that TDD aided developers in producing more solid and dependable code. Thisis most likely a result of the safety net that TDD offers, which catches mistakes early in the development process and enables developers to rectify them before they become significant problems. Over time, this results in code that is more dependable and simpler to maintain.

In addition, TDD improved developers' understanding of the software's requirements, according to our study. Writing tests first forces developers to consider the needs of the software more carefully, resulting in a more precise and thorough implementation of those needs. [10]

Overall, our case study shows that TDD is a powerful software development technique for raising the caliber of software. The findings of our survey show that TDD can aid in better understanding software requirements, writing more robust and dependable code, and identifying and fixing flaws sooner in the development process. These advantages may result in a completed product with fewer flaws, lower upkeep costs, and more client satisfaction.

V. CONCLUSION

As a result, our case study offers convincing proof that Test-Driven Development (TDD) is an effective strategy for raising the caliber of software. According to our research, TDD enabled developers to find and correct bugs earlier in the development cycle, leading to a final product with fewer errors. As a result, software development projects may see cheaper maintenance costs, higher customer satisfaction, and ultimately, a better return on investment. [10]

Additionally, TDD provided a safety net that catches errors early in the development process, assisting developers in writing more durable and dependable code. TDD also aided developers in having a better understanding of the software's requirements, which resulted in a more precise and thorough implementation of those requirements.

Overall, TDD offers many benefits for software development projects and its impact on software quality cannot be overstated. Our case study highlights the importance of adopting TDD in software development projects to ensure that the final product meets the highest standards of quality and reliability.

REFERENCES

[1].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, pp. 45–54, 2016. [Online]. Available:https://www.sciencedirect.com/science/article/pii/S0950584916300222

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

[3].T. Freese, "Towards software configuration management for test-driven development," in *Software Configuration Management*, B. Westfechtel and A. van der Hoek, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 267–273.

[4].J. Spacco and W. Pugh, "Helping students appreciate test-driven development (tdd)," in *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*, ser. OOPSLA '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 907–913. [Online]. Available: https://doi.org/10.1145/1176617.1176743

[5].G. Canfora, A. Cimitile, F. Garcia, M. Piattini, and C. A. Visaggio, "Evaluating advantages of test-driven development: a controlled ex- periment with professionals," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, 2006, pp. 364–371.

[6].George and L. Williams, "A structured experiment of test-driven development," *Information and Software Technology*, vol. 46, no. 5, pp. 337–342, 2004, Special Issue on Software Engineering, Applications, Practices and Tools from the ACM Symposium on Applied Computing 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584903002040

[7].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)*, 2007, pp. 275–284.

[8].Desai, D. Janzen, and K. Savage, "A survey of evidence for test-driven development in academia," *SIGCSE Bull.*, vol. 40, no. 2, p. 97–101, Jun 2008. [Online].Available: https://doi.org/10.1145/1383602.1383644

[9].Janzen and H. Saiedian, "Does test-drive development really im- prove software design quality?" *IEEE Software*, vol. 25, no. 2, pp. 77– 84, 2008.