Comparison between Test-Driven Development and Conventional Development: A Case Study

Norah AlHammad
Tamkeen Technologies
Riyadh,Saudi Arabia
norah.a.alhammad@gmail.com

Arwa AlKowiter,Nahed AlSahdi & Taghreed AlOtaibi
Prince Sultan University
Riyadh,Saudi Arabia
arwa1990.aa@gmail.com,nahed-al9hdi@live.com & taghreedd@live.com

Lujain AlThunayan
Hewlett Packard Enterprise
Riyadh,Saudi Arabia
lujain.th@gmail.com

***Abstract*— In Software Engineering, different techniques and approaches are being used nowadays to produce reliable software. The software quality relies heavily on the software testing. However, not all developers are concerned with the testing stage of a software. This has affected the software quality and has increased the cost as well. To avoid these issues, researchers paid a lot of effort on finding the best technique that guarantee the software quality. In this paper we aim to explore the effectiveness of building test cases using Test-Driven Development (TDD) technique compared with the conventional technique (Test-last). The comparison measures the effectiveness of test cases with regard to number of defects, code coverage and test cases development duration between TDD and Test-Last. The results has been analyzes and presented to support the best technique. On an average, the effectiveness of test cases with regards to the selected quality factors in Test-Driven Development (TDD) was better than the conventional technique (Test-last). TDD and conventional testing had nearly the same percentage as result in code coverage. Moreover, the number of defects found and the test cases development duration spent in TDD are high compared with Test-Last. The results led to suggest some contributions and achievement that could be gained from applying TDD technique in software industry. As using TDD as development technique in young companies can produce high quality software in less time.**

***IndexTerms*—Test driven development, Conventional Development, Quality factors, Software Development Styles, Unit Testing, Experimental Analysis.**

I. Introduction

In today's fast moving world, the competition between software development companies has increased and customers tend to head to companies that provide reliable and high software quality. Excellent software should be completely well tested before the final release. However, most developers are postponing test activities until the end of the development process. This postponing increased the overall cost of the software. It also affected the software quality in many levels. There were many researches and experiments that have been conducted to address the mentioned issues. The focus of many researches is to find the best practice to discover errors and defects. Although there are many testing techniques and methods are found, two techniques have been selected for this case study: Test-Driven Development TDD (Test-First) and conventional testing technique (Test-Last).

The most and well-known testing methodology used is the conventional testing technique (Test-Last), where testing is done after developing the code. In Test-Last, the developer ensures the code is working properly as required. Test cases are built then to improve the code and ensure its correctness.

On the other hand, TDD was first introduced by Kent Beck as a development technique, which is considered as a part of the software development agile methodologies [1]. In TDD, test cases are created based on the customer requirements. Developers start writing an automated unit test case before writing any line of code, and then they execute the test case, which will fail at first. Then, developers start refactoring the code by adding the required methods and functionalities until they pass the test cases. This mentioned cycle goes on to cover all the functionalities that are being implemented [2].

In software testing, researchers paid a lot of effort over which testing technique is the most effective for the software. In conventional technique (Test-Last), the code correctness is a high priority. Therefore, it is certain that the requirements are fulfilled as required. However, it does not guarantee the discovery of all expected defects in the code. On the other hand, In TDD the implementation of test cases in advance with the required code for each test case has many benefits. It helps the developers to protect the implemented features during code refactoring. Moreover, it is considered helpful in obtaining a regular feedback even after any change is made (either because of changing user requirements or the change generated from developer's side). Moreover, it helps to reduce overall cost and increase the quality of test cases. However, TDD requires an experience and high skilled developer in order to build efficient and effective test cases. Moreover, TDD consumes time when it is applied on small software.

In this paper we explore the effectiveness of building test cases using Test-Driven Development (TDD) technique compared with the conventional technique (Test-last). The comparison will measure the effectiveness with regard to number of defects, code coverage and test cases development duration between TDD and Test-Last. The results are analyzed to support the best technique. Moreover, the outcomes will help the developer in choosing the best practice in unit testing.

The reminder of this paper is structured as follow. Section II describes the literature review of papers that were explored during this research. Section III describes the methodology that

IRJECE

was followed during conducting this experiment. Section IV presents the case study analysis and the results discussion. Finally, Section V presents conclusion and future work.

II. Background

In software development cycle, software testing is one of the most important time consuming step. Unit testing is a software testing method by which separate units of source code are tested to determine whether they are usable or not. There are many techniques and approaches for software testing. The conventional testing technique is writing test cases after completing the full implementation. This is mainly to verify the code is working properly. In the other hand, TDD technique is writing test cases before writing the code. Therefore, the only key difference between the two techniques is the stage of testing; Test-First in TDD and Test-Last in conventional testing technique.

Recently, researches have started to conduct studies on the effectiveness of TDD technique compared with conventional technique. Erdogmus and Morisio [3] had conducted a controlled experiment with undergraduate students for evaluating the important of TDD. The students were asked to implement a small functionality; a group had applied a test-first strategy, where the other group had applied the test-last strategy. They have found that the test-first students were more productive due to the many test cases that were written. They believe that applying the test-first approach increases the requirement understanding, reduces the scope of the task to be performed, reduces debugging and rework effort and achieved more consistent quality results.

Gupta and Jalote [4] had evaluated the effectiveness and efficiency of TDD compared to conventional technique by conducting an experiment on two groups of students. Both groups were asked to develop a medium sized program, where one group follows the TDD approach, and the other follows the conventional way for developing a program. The results of the study showed that TDD approach is more efficient as it required less development effort and more productivity for the developers. The code quality was affected too by the testing effort applied by using TDD. They have noticed that the developers may prefer a modified TDD in which some upfront designing is done before developing the code using TDD.

Another structured experiment was conducted by Muller and Hagner [5] to compare TDD with traditional programming on 19 students. The main purpose was to measure the effectiveness of TDD in terms of development time, program reliability and understandability. They have observed that TDD programmers reuse existing methods fast and accurate, as well as an increased reliability. However, the programming time was neither efficient nor faster as expected.

Williams, Maximillien and Vouk [6] have run a case study at IBM where they have transitioned from an adhoc technique to a TDD unit testing technique. They have found that the developed code using TDD during functional verification and regression test, showed approximately 40% fewer defects in the code comparing to the code developed in a more traditional technique. Moreover the TDD technique will support the future enhancements, reusability, maintenance and the quality of the code. However, the productivity was not impacted by the focus on producing automated test cases. As TDD is an advance technique that uses an automated test cases that can be run at any time, to ensure that the software is still working properly.

Moreover, Janzen and Saiedian [7] have conducted a study to collect evidence regarding the TDD influence on software. They have found that developers implementing TDD tend to write software in smaller less complex units and well tested. In the other hand, Causevic, Sundmark and Punnekkat [8] have conducted a systematic literature review on empirical studies focusing on TDD limitations. They have identified several limitations such as increased development time, lack of TDD experience/knowledge and lack of developers' skills in writing test cases.

III. Methodology

We conduct a case study where two programmers are asked to develop and test some functions related to objected-oriented programming. The first participant is responsible to test these functions by applying the TDD technique. Where the other participant is responsible to test the same functions by applying the conventional technique (Test-Last), which creates test cases after developing the code. The main focus of this experiment is to compare some quality attributes such as: number of defects, code coverage and test cases development duration between TDD with the conventional technique, and document the results and outcomes of the comparison. This section describes how the case study is designed and applied. Moreover, specific details regarding the case study are explained below.

A. Quality

In this study, we have selected specific quality factors to evaluate. These factors play an important role in the effectiveness of the test cases. Also, they are considered significant aspects in the software development life cycle. The software quality can be measured using many different attributes. The selected quality factors are:

a) Number of defects

An indicator of the defects number found in each function.

b) Code Coverage

A measure used to describe how much the source code is covered and tested by a particular test case.

c) Duration

A measure of the time spent to develop the test cases.

B. Environment and Tools

In this case study, we use an object oriented programming language (Java) to develop the functions using Eclipse [9]. Eclipse is a Java based open source platform that allows a software developer to create an integrated development environment (IDE). In order to write and execute test cases we use JUnit [10]. JUnit is an open source framework that has a graphical user interface (GUI), it shows a test progress bar in Eclipse. After completely finalizing the implementation, we install a tool called EclEmma [11] for code coverage. EclEmma is a free Java code coverage tool for Eclipse that used as an indicator on how much the tests case cover the source code for each function.

IRJECE

C. Experiment Design

The study has been applied by two participants; the first participant is a software developer with a three years' experience. The second participant is a services integration tester with a three years' experience. Each participant is assigned to implement and test the same specific functions. These specific functions are used to calculate statistics formulas, which are: Mean, Median and Standard Deviation. The first participant is responsible to test and develop each function using TDD technique. Where the other participant is responsible to develop and test the same functions using Test-Last technique. During this experiment, each participant has developed and implemented the test cases with regard to the quality factors mentioned in section (A) above. The results are then collected and analyzed for each quality factor. Finally, we have presented our viewpoint and interpretation from the case study. "Figure 1"illustrate the methodology used during the case study.

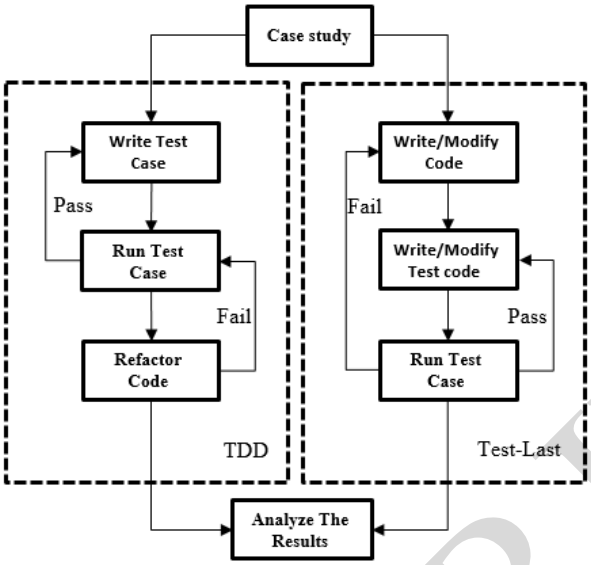


Fig. 1. Methodology used in the case study

IV. RESULTS AND DISCUSSION

A. Results Analysis

The results are viewed for both TDD and Test-Last for each function with regard to each quality factor.

Table I represents the number of test cases generated for each function by each participant. The first participant developed 14 test cases, which is more than the second participant that develop only 4 test cases. The reason behind the difference is that the first participant, who followed TDD technique, was writing test case first, which drove the participant to write more code and run the tests for it. So, many test cases kept rising during writing the code. The first participant was forced to think as a tester and develop the function correctly accordingly. As for the second participant, who followed the Test-Last technique, started writing the test cases after finishing the functions' development. The second participant's focus was only to confirm each function is working properly and that it is doing what it is required to.

TABLE I. NUMBER OF TEST CASES

	Participant 1 Test Driven Development (TDD)	**Participant 2 Test-Last**
	Number of test cases	***Number of test cases***
Function 1 (Mean)	5 Test Cases	1 Test Cases
Function 2 (Median)	4 Test Cases	2 Test Cases
Function 3 (Standard Deviation)	5 Test Cases	1 Test Cases
Total	**14**	**4**

Table II lists the duration time needed for each participant to fully implement the functions. The duration includes both testing and coding time. The first participant consumed 192 minutes to complete all the functions, which is considered higher than the second participant that consumed 144 minutes for all the functions. The reason behind the variance is that the number of test cases generated for each function was totally different. For example, the first function in TDD, 5 test cases took 63 minutes to complete the function. While in Test-Last, it took 37 minutes to develop the function and the test case. Therefore, the time variance between the two participants was based on number of test cases, and the time spent handling each case.

TABLE II. DURATION OF DEVELOPMENT TIME FOR EACH TEST CASE

	Participant 1 Test Driven Development (TDD)		**Participant 2 Test-Last**		
	Test Case	***Time***	***Coding time***	***Test Case***	***Testing Time***
Function 1 (Mean)	*TC1*	11 minutes	20 minutes	TC1	17 minutes
	TC2	14 minutes			
	TC3	15 minutes			
	TC4	13minutes			
	TC5	10 minutes			
Function 2 (Median)	*TC1*	17 minutes	40 minutes	TC1	12 minutes
	TC2	13 minutes			
	TC3	10 minutes		TC2	10 minutes
	TC4	15 minutes			
Function 3 (Standard Deviation)	*TC1*	20 minutes	30 minutes	TC1	15 minutes
	TC2	11 minutes			
	TC3	15 minutes			
	TC4	17 minutes			
	TC5	11 minutes			
Total	**192 minutes**		**144 minutes**		

Table III lists the total number of failing assertions found in each test case for each participant. The total number of defects discovers by the first participant are 56 defects. Where the second participant discovered 21 defects. If we look at function 2 for the first participant, there were a total of 14 failed assertions for the 4 test cases during the code. However, the other participant detected 8 failed assertions for the 2 test cases. Clearly, the more test cases generated, the more defects are being detected.

IRJECE

TABLE III. NUMBER OF DEFECTS FOR EACH TEST CASE

	Participant 1 Test Driven Development (TDD)		Participant 2 Test-Last	
	Test case	*Defects*	*Test case*	*Defects*
Mean	TC1	2	TC1	5
	TC2	5		
	TC3	5		
	TC4	3		
	TC5	2		
Median	TC1	5	TC1	4
	TC2	3		
	TC3	2	TC2	4
	TC4	4		
Standard Deviation	TC1	8	TC1	8
	TC2	3		
	TC3	4		
	TC4	7		
	TC5	3		
Total	**56 defects**		**21 defects**	

Table IV lists the code coverage (score in percentage) for covered instructions and missed instructions for each participant. Code coverage is calculated by dividing the covered instructions by the summation of the covered instructions and missed instructions. The result is multiplied by 100 to calculate the percentage. Below is the formula for code coverage: Code coverage in % = [Covered instructions / (Covered instructions+ Missed instructions)]* 100 we have found that the coverage that was achieved by both participants is nearly the same, regardless of the technique they have used. This makes it difficult to distinct a difference in the effectiveness between the two techniques.

TABLE IV. CODE COVERAGE FOR EACH TEST CASE

	Participant 1 Test Driven Development (TDD)				Participant 2 Test-Last			
	Test case	*Code coverage in %*	*Covered instructions*	*Missed instructions*	*Test case*	*Code coverage in %*	*Covered instructions*	*Missed instructions*
Mean	TC1	95.45 %	21	1	TC1	97.30 %	36	1
	TC2	100%	53	0				
	TC3	100%	53	0				
	TC4	100%	43	0				
	TC5	95%	19	1				
Total	**98.95%**		**189**	**2**	**Total**	**97.30 %**	**36**	**1**
Median	TC1	97.67 %	42	1	TC1	93.62 %	44	3
	TC2	100%	23	0				
	TC3	100%	21	0	TC2	100%	42	0
	TC4	97.92 %	47	1				
Total	**98.52%**		**133**	**2**	**Total**	**96.63 %**	**86**	**3**
Standard Deviation	TC1	95.35 %	41	2	TC1	95.24 %	40	2
	TC2	100%	21	0				
	TC3	100%	43	0				
	TC4	97.67 %	42	1				
	TC5	100%	23	0				
Total	**98.27%**		**170**	**3**	**Total**	**95.24 %**	**40**	**2**

B. Interpretation

As a result of comparing Test-Driven-Development technique with the conventional technique Test-Last, several remarks have been observed. The participant who applied the conventional technique (Test-Last) has reported some points. One is that applying Test-Last has ensured that the requirements for each function are well understood by the participant before testing has started. In addition, the participant was focusing on ensuring the correctness of code before moving to testing stage. Therefore, the creditability of the functions is guaranteed first, then test cases can be built and code can be improved accordingly.

On the other hand, the participant who applied TDD technique has noticed many changes too during the development. One is that TDD has forced the developer to simplify the code and write code based on the requirement of the tests only. In addition, the participant who applied TDD was developing the function by following clear steps, since writing the code is done during the testing. Moreover, since test cases are written before the code, code coverage is ensured at least one for each function. Moreover, we have noticed that the number of defects and test cases development duration spent in TDD are considered high compared with Test-Last. However, in the future when a change is made to the software, all what the developer has to do is run the existing test cases to test if the change has impacted any other pieces of the software. Hence, for the long term it could take less time in coding and debugging, and less defects might be found for any future integration or any change. Therefore, this finding could support software maintainability, which is one of the most remarks of TDD that it is considered beneficial in future is software maintainability. The reason is that when the software has an update, it could be done easily by just developing a new test case. This assures the ability of making changes fast without interfering any other piece of the software. This could help young companies whose looking up for rapid growth. As using TDD as development technique in young companies can produce high quality software in less time. However, during conducting the experiment, it has been noticed that TDD requires an experience and high skills on building test cases before the code. So beginner programmers might face difficulties with TDD compared with conventional technique.

V. CONCLUSION AND FUTURE WORK

In software development cycle, software testing is one of the most important time consuming step. Unit testing is a software testing method by which separate units of source code are tested to determine whether they are usable or not. The most well known method is the conventional technique (Test-Last), where building test cases are done after developing the code. Another method is Test-Driven Development (TDD), which is one of the most advanced techniques and mostly known in agile methodologies. In TDD, Test cases are built before and during code development. In this paper, we have explored the effectiveness of building test cases using Test-Driven Development (TDD) technique compared with the conventional technique (Test-last). The comparison measures

IRJECE

the effectiveness with regard to number of defects, code coverage and test cases development duration between TDD and Test-Last. The aim is to determine the best technique and approach when developing software with regards to effectiveness of test cases. The main findings of the analysis are the following: we have found that TDD and conventional testing had nearly the same percentage as result in code coverage. Moreover, the number of defects found and the test cases development duration spent in TDD are high compared with Test-Last. However, for the long term it could take less time and less defects might be found for any future integration or any change. However, TDD requires more experience and has less code correctness compared with conventional technique.

In a future work for a long-term study, it is recommended to expand the experiment to cover more participants and apply it on a large system or a web application. Moreover, it will be more useful to conduct a fully controlled experiment with higher number of participant in order to investigate the effectiveness of TDD with integration and validation testing. This is to help long-term organizational plans to reduce the effort and cost.

ACKNOWLEDGMENT

We are greatly thankful to the faculty of Software Engineering department in Prince Sultan University, Especially Dr. Sharifa Al-Ghowinem for her guidance, valuable comments, and support in overcoming the hurdles in the completion of this research.

At last but not the least, we also thank Ms. Ghada Hader for her co-operation and encouragement in successfully completing this research.

REFERENCES

[1] Agarwal, N and Deep, P., "Obtaining Better Software Product by Using Test First Programming Technique", 5th International Conference on Confluence The Next Generation Information Technology Summit (Confluence), 2014, Pages: 742 – 747 [Online]. Available: http://ieeexplore.ieee.org.ezproxy.psu.edu.sa/stamp/stamp.jsp?tp=&arnumber=6949233

[2] Karamat, T and Jamil, A.N., "Reducing Test Cost and Improving Documentation In TDD (Test Driven Development)", Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing-, 2006, Pages: 73 – 76, [Online]. Available: http://ieeexplore.ieee.org.ezproxy.psu.edu.sa/stamp/stamp.jsp?tp=&arnumber=1640669

[3] Erdogmus, H.; Morisio, Maurizio; Torchiano, Marco, "On the Effectiveness of the Test-First Approach to Programming", Software Engineering, IEEE Transactions, 2005, Pages: 226 - 237, [Online]. Available: http://ieeexplore.ieee.org.ezproxy.psu.edu.sa/stamp/stamp.jsp?tp=&arnumber=1423994

[4] Gupta, A.; Jalote, P., "An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development", First International Symposium on Empirical Software Engineering and Measurement, 2007, Pages: 285 - 294, [Online]. Available: http://ieeexplore.ieee.org.ezproxy.psu.edu.sa/stamp/stamp.jsp?tp=&arnumber=1423994

[5] Muller, M.M.; Hagner, O., "Experiment about test-first programming", 2002, Pages: 131- 136, [Online]. Available: http://ieeexplore.ieee.org.ezproxy.psu.edu.sa/stamp/stamp.jsp?tp=&arnumber=1049202

[6] Williams, L.; Maximilien, E.M.; Vouk, M., "Test-Driven Development as a Defect-Reduction Practice", 14th International Symposium on Software Reliability Engineering, 2003, Pages: 34 - 45, [Online]. Available: http://ieeexplore.ieee.org.ezproxy.psu.edu.sa/stamp/stamp.jsp?tp=&arnumber=1251029

[7] Janzen, D.S.; Saiedian, H., "Does Test-Driven Development Really Improve Software Design Quality?", 2003, Pages: 34 - 45, [Online]. Available: http://ieeexplore.ieee.org.ezproxy.psu.edu.sa/stamp/stamp.jsp?tp=&arnumber=4455636

[8] Causevic, A.; Sundmark, Daniel; Punnekkat, S., "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review", IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST), 2011,Pages: 337 - 346, [Online]. Available: http://ieeexplore.ieee.org.ezproxy.psu.edu.sa/stamp/stamp.jsp?tp=&arnumber=5770623

[9] Eclipse, http://www.eclipse.org

[10] jUnit Framework, http://www.junit.org

[11] EclEmma - Java Code Coverage for Eclipse, http://www.eclemma.org

IRJECE