Consequences of Dependent and Independent Variables based on Acceptance Test Suite Metric Using Test Driven Development Approach

1st Myint Myint Moe
Faculty of Information Science
University of Computer Studies (Hpa-an),
Kayin State, Myanmar
myintmyintmoe.ucsy.1971@gmail.com

2nd Khine Khine Oo
Faculty of Information Science
University of Computer Studies, Yangon
Myanmar
khinekhineoo@ucsy.edu.mm

Abstract

The fundamental of software development was Test-Driven Development but the individual tests must be carried out previously the production code. To research, the consequence of test-driven development on product code quality and developer productivity was the destination of this paper. This system builds the acceptance test suite metric of regression analysis to assess the impact of the process on dependent variables and independent variables. This paper's results observed the positive effect of external quality over function of the number tests, and slightly decrease the effect of developer productivity over function of the number of tests. TDD can affect advance software products' quality, also mend programmers' productivity. TDD undertook to help the delivery of high-quality products, both operational (fewer bugs) and technical perspective (cleaner code) while improving developers' productivity. TDD affects to less defects and fewer debugging period which correct code can be certified by writing tests first and thus serving the developer get a finer understanding of the software requirements.

Keywords-- *Test-Driven development, Unit test, no: of tests, External Quality, Developer Productivity*

I. INTRODUCTION

By driving from Extreme Programming (XP) and the primary of the Agile Platform, the foundation fragment of the agile code development approach was Test-driven development (TDD). The possibility of TDD describes various positive effects. TDD isn't a testing approach, yet rather a development and design method in which the tests are composed before the production code. During the implementation, the tests are added step by step and when the test is passed, the code is refactored to improve the inside structure of the code, without changing its outside behavior. TDD cycle is iterated until the whole functionality is performed. An automated segment of code was a unit test that applied a part of work in the system and a unique idea about the execution of that part of work. For each little function of an application, TDD begins with designing and developing tests. First, the test is created that distinguishes and approves what the code will do in the TDD approach. Make the code and after that test in the typical testing process. The developer can be self- assurance that code refactoring is not destroyed any existing functionality for re-executing the test cases. Before the actual development of the application, TDD is a process of evolving and running automated tests. To create higher code quality, developers can motivate by coding standards, analyzing code automatically, doing code reviews and refactoring legacy code. For bugs and defects count, the system works by testing and debugging.

This paper is structured as follows. The issue of Test-Driven Development initiated in Section (1). The obviousness of the test numbers, quality of external code and, developer product on test-driven development (TDD) expressed in Section (2). Section (3) describes related work. Section (4) presents a framework of test-driven development. The contribution of the relation of the test numbers, quality of external code, and developer product is described in Section (5). Next, observational analysis of the proposed system is discussed in Section (6). Section (7) expresses compatibility of results. Finally, Section (8) concludes this paper.

II. OBJECTIVES

One of the approaches of software progression was test-driven development. In recent years, this approach has become familiar in the industry as a requirements specification method. Before code development, developers encourage to compose tests. TDD is provided to carry the code clearer, simple and

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

bug-free. The goal of the proposed system analyzes the consequence of dependent variables and independent variables on TDD**.** It observes the nature of the interaction between test numbers (TEST), quality of external code (QLTY), and the relation between the test numbers (TEST) and developers' product (PROD). This decreases the fault of enhanced software either instantly or in the long run. The benefits of TDD enhanced software quality and speed up the testing process. This approach aims more productive and make fewer efforts per line of code. By decreasing code complexity supporting, the proposed system validates the exactness of all codes and allows developers assurance. It is used persistently over time and motivates developers to create higher code quality.

III. RELATED WORK

In [5], Authors (Y. Rafique and V. B Misic) described on "The effects of test-driven development on external quality and productivity: A meta-analysis". Authors reported that TDD improves external quality when compared to a waterfall approach. However, this improvement is not strong. Further, TDD becomes disadvantageous for the subset containing only academic studies in which it is compared to an iterative, test-last (ITL) process instead of a waterfall approach. This result suggests that sequencing might have a negative effect on external quality, which we haven't observed. Productivity results are more inconclusive in that the authors report a small productivity hit for TDD when comparing TDD with waterfall but the effect, even though still small, is reserved when ITL is compared with TDD.

In [14], Authors (H. Munir, K. Wnuk, K. Petersen, and M. Moayyed) proposed on "An experimental evaluation of test-driven development vs. test-last development with industry professionals". The authors were developed that it intended to compare the effect performed by TDD and TLD (Test-Last Development) on the quality of internal and external code, and developer's product. For this aim, 7 user stories' a programming exercise was carried out. The results of the analysis by the approved test cases' number: McCabe's Cyclomatic complexity, branch coverage, the no: of code lines person/hour, and user stories' number described person/ hour. The tests expressed fewer significant enhancements in accept of TDD, by reducing the defects. In terms of productivity, the tests indicate that TDD than TLD slightly decrease average productivity.

In [15], Authors (M. Moayyed, H. Munir, and K. Petersen) described on "Considering rigor and relevance when evaluating test-driven development: A systematic review". Authors were developed that the primary studies are considered together; however, the nine better-rigors, better-relevance studies describe that TDD enhances quality of external code, while developer' product is not effected. The 21 studies in the alike classification of the basis analyze and this replication are ambiguous both results.

IV. BACKGROUND THEORY

Test-Driven Development is a coding technique. TDD accelerates the before time development of tests, at the time alternates are accepted and improved with functional components. Kent Beck invents Test-Driven Development applies to a form of programming although three actions are exactly interlinked. Three activities are Coding, Testing, and Design. At first, its key idea is to execute early initial tests for the code, must be actualized but the accurate feature of it used. One of the features of software system requirement is tackled subtask or user stories, which are designed to easily express and understand. These can be easy to change by the end-user as they like during the project's handle time.

A. Test-Driven Development

The TDD process is expressed in Figure 1, and includes the consecutively steps:

1. Write only one test-case
2. Run or perform this test-case. If this test-case fails, go to step 3. If the test-case succeeds, go to step 1.
3. Refactor the performance to get the elementary design possible.
4. Enable the minimal code to do the test-case run.
5. Run the test-case again. If it fails again, go to step 3. If the test-case succeeds, go to step 5.
6. Again, run the test-case, to certify that the refactored application until passes the test-case. If the test-case fails, go to step 3. If the test-case passes, go to step 1, if there are still requirements, left in the specification.

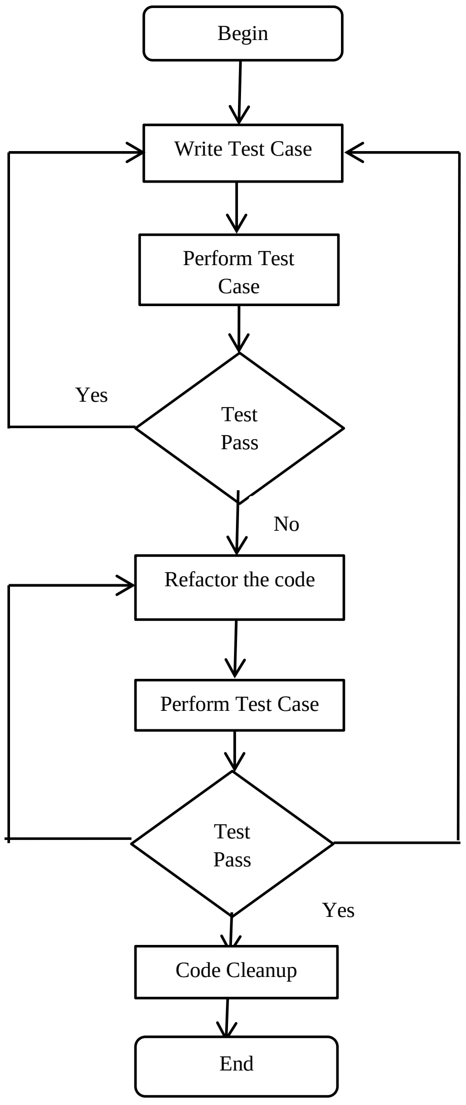

Figure 1: Test-Driven Development flow

V. CONTRIBUTION

First stage, the original study of Test-Driven Development has beneficial effects on the number of unit-test written by the developers, the external code quality and the developers' productivity. In the second stage, the authors studied the correlation between the number of tests, the external code quality, and productivity. TDD approach encourages developers to write more tests and is a positive correlation between the number of tests, quality, and productivity, and then TDD would improve the overall quality and productivity. The related work observed, if code quality has a positive effect, productivity has a negative effect, if productivity has a positive effect, code quality has a negative effect. The proposed work discovered, if code quality has a positive effect, productivity has a slightly decrease effect, if productivity has a positive effect, code quality has fewer reduced effect.

VI. PROPOSED SYSTEM

In this proposed method the acceptance test suite metric of regression analysis uses to measure the no: of test numbers, quality of external code, and developer product.

A. Research Questions

This system concentrates to evaluate two outcomes on the following system: external code quality and developer productivity.

RQ1 (RQ-QLTY): Does a higher number of tests indicate higher quality?

RQ2 (RQ-PROD): Does a higher number of tests indicate higher developer productivity?

The notion of external code quality in RQ-QLTY and productivity in RQ-PROD are based on the acceptance test suite metric of regression analysis.

B. Method

In the proposed system, the acceptance test suite metric is used by analyzing to explore possible interactions such as number of tests, external code quality, and developer productivity. The acceptance test suite metric is a form of mathematical regression analysis. Regression analysis is used to investigate the relationship between two or more variables and estimate one variable based on the others. Regression analysis is a powerful statistical method that allows for analyzing the relationship between two or more outcome variables of interest. QLTY and PROD are the dependent variables. TEST is the independent variable. QLTY defined as the percentage of acceptance tests passed for the implemented tackled tasks. PROD measured as the percentage of implemented tackled tasks. Table 1 provides the raw data used in the assessment. To compute this low-level measure, an automated tool used by this system. The limited-time necessary to complete the task had an impact on the metric. In regression analysis, dependent variables are established on the vertical y-axis, while independent variables are established on the horizontal x-axis.

C. Test Numbers (TEST)

Test numbers (TEST) is identified as JUnit assert statement numbers inside the unit test suite written by the participants while tackling the task. The numbers of test development as a single JUnit assert statements. TEST assessed by the count of the JUnit test cases. TEST is a ratio measure in the range [0, ∞]. The formula for calculating TEST is defined as [10]:

TEST = no: of subtasks out of result the no: of input subtasks (1)

TEST = JUnit assert statement numbers inside the unit test suite

Table 1: Summary of acceptance tests used to calculate the metrics of Bowling Scorekeeper data-sets [7].

Task	Test	Assert
T1	3	3
T2	3	3
T3	2	2
T4	3	10
T5	5	5
T6	6	6
T7	8	8
T8	5	5
T9	5	5
T10	4	4
T11	2	2
T12	3	3
T13	2	2

D. External code quality

The metric for external quality QLTY based on the number of tackled subtasks *(#TST*) for a given task. A subtask as tackled assesses if at least one assert statement in the acceptance test suite associated with that subtask passes. QLTY is a proportion measure in the range 0 to 100.

The number of tackled subtasks (*#TST*) is defined as:

$$\#TST = \sum_{i=0}^{n} \begin{cases} 1 & Assert_i(Pass) > 0 \\ 0 & otherwise \end{cases} \quad (2)$$

#TST = the number of tackled subtasks

n = the total number of subtasks

The formula for measuring QLTY is defined as [8]:

$$\text{QLTY} = \frac{\sum_{i=0}^{\#TST} QLTY_i}{\#TST} \times 100 \quad (3)$$

QLTY_i = the ith tackled subtask's quality

Where QLTY_i is the quality of the i^{th} tackled subtask and QLTY_i is defined as:

$$\text{QLTY}_i = \frac{\#Assert_i(Pass)}{\#Assert_i(All)} \quad (4)$$

$\#Assert_i$ (Pass) = the number of JUnit assertions passing in the acceptance test suite associated with the ith subtask

$\#Assert_i$ (All) = the total number of JUnit assertions in the acceptance test suite associated with the ith subtask

For example, supposing that the thirteen tackled subtasks (*#TST* = 13) assessed by a person, this denotes that the thirteen tackled subtasks pass more than one assert statement in the test suite. Assume us that the acceptance test of the first analyzed tackled task contains 3 assertions, out of results of three are passing. The acceptance tests of the fourth tackled task contain 10 assertions, out of results of three are passing and so on.

Table 2: Solution of QLTY

Task	Test	Assert	QLTY
T1	3	3	1
T2	3	3	1
T3	2	2	1
T4	3	10	0.3
T5	5	5	1
T6	6	6	1
T7	8	8	1
T8	5	5	1
T9	5	5	1
T10	4	4	1
T11	2	2	1
T12	3	3	1
T13	2	2	1
	51	**58**	**95**

i.e. $(\text{QLTY}_4 = \frac{\#Assert_4(Pass)}{\#ASSERT_4(All)} = \frac{3}{10} = 0.3)$

$$(\text{QLTY} = \frac{\sum_{i=1}^{\#TST} QLTYi}{\#TST} \times 100$$

$$= \frac{1+1+1+0.3+1+1+1+1+1+1+1+1+1}{13} \times 100 = 95)$$

E. Productivity

The productivity metric (PROD) expresses the amount of work effectively carried out by the subjects. PROD is a proportion measure in the range 0 to100. The metric of PROD is computed as follows [8]:

$$\text{PROD} = \frac{\#Assert(Pass)}{\#Assert(All)} \times 100 \qquad (5)$$

For sample, assume a tacked task with all of 58 assert statements enabled by a person in a test suite. After compiling, the person's outcome 51 asserts statements are passing.

i.e. $(\text{PROD} = \frac{\#Assert(Pass)}{\#Assert(All)} \times 100 = \frac{51}{58} \times 100 = 88)$

F. Assessment

The image below is a scatter plot**.** Scatter plots are used when this paper want to show the relationship between two variables. Scatter plots are known as relationship plots because they show how two variables are interrelated. This analytical tool is most often applied to show data correlation between two variables. This system expects that the regression assessment of the tackled task compiled from the quality of external code on test numbers by TDD responds positively to questions RQ1. In the same way, this system expects that the regression analysis of the tackled task compiled from the developer product on test numbers by TDD responds slightly decrease to questions RQ2.

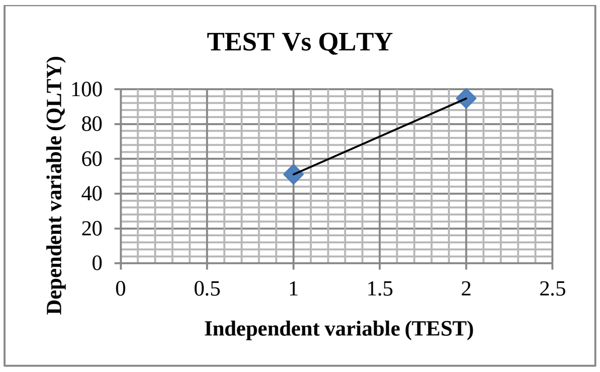

Figure 2: QLTY is on the function of TEST

In figure 2, the external code quality on the test numbers is improved by measuring the acceptance test suite metric of quality (QLTY).

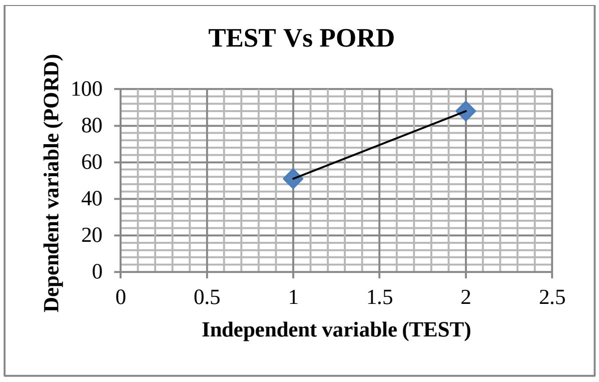

Figure 3: PROD is on the function of TEST

In figure 3, the developer's productivity over the test numbers is slightly decreased by measuring the acceptance test suite metric of productivity (PROD).

VII. COMPATIBILITY OF RESULT

In this portion, this paper presents the outcomes acceptance test suite metric of regression analysis. Further, a significant relation between TEST and QLTY, as expressed in *RQ1*, with a positive was found. Hence, scatter plot figure 2 is an arithmetically expressive relationship between the number of tests and external code quality. Additionally, a significant relation between *TEST* and PORD, as expressed in *RQ2*, with a somewhat down was found. So, scatterplot figure 3 is an arithmetically expressive correlation between the number of tests and programmer productivity. In this study, the number of tests is a good predictor for TDD programmer productivity. Consequently, developer product on the test numbers becomes lightly diminishment and quality of external code on the test numbers becomes improvement. Development time is relatively high in the proposed work. It nearly takes as much as 16% more time than that of related work. Proposed work decreases the maintenance cost and overall increases the productivity. There are as many as 52% more test cases as of the related work. The related work codes have a relatively small size. The inclusion of many more test cases in the proposed work increases the size of the code. The related work codes are simpler. The cyclomatic complexity of the related work is relatively smaller. The proposed work is relatively complex.

VIII. CONCLUSION

Development time is relatively high in the proposed work. It nearly takes as much as 16% more time than that of related work. Proposed work decreases the maintenance cost and overall increases the productivity. There are as many as 52% more test cases as of the related work. The related work codes have a relatively small size. The inclusion of many more test cases in the proposed work increases the size of the code. The related work codes are simpler. The cyclomatic complexity of the related work is relatively smaller. The proposed work is relatively complex.

This approach allows thorough unit testing which enhances the quality of the software and advances customer satisfaction. They help with retaining and varying the code. Moreover, the number of acceptance test cases passed and number defects found through static code analysis are used to measure the external code quality. All these measures are consistent with the studies and will be considered as standard measures. When this proposed system assesses the acceptance test suite metric of regression analysis, the result of developer productivity over the number of tests is fewer decreased and the result of external code quality over the number of tests is increased in giving a fixed time-frame.

ACKNOWLEDGMENTS

This research paper is partially supported by academic studies. Professionals were fit to implement more effective with test-driven development. Furthermore, this proposed system observes that the measurement reveal different aspects of a development approach in academic studies.

REFERENCES

[1] Causineou and Chartier, 2010; Outliers Detection and Treatment: a Review, International Journal of Psychological Research, 3(1): 58-67.}

[2] H. Kou, P. M. Johnson, and H. Erdogmus, “Operational definition and automated inference of test-driven development with Zorro,” Automated Software Engineering, 2010.

[3] Shaweta Kumar, Sanjeev bansal, “Comparative Study of Test driven Development with Traditional Techniques”; International Journal of Soft computing and Engineering (IJSCE); ISSN:2231-2307,Volume-3, Issue-1, (March 2013).

[4] A.N. Seshu Kumar and S. Vasavi ; “Effective Unit Testing Framework for Automation of Windows Applications”; Aswatha Kumar M.et al.(Eds); Proceedings of ICADC, AISC 174, pp. 813-822. Springerlink .com @ Springer India 2013

[5] Y. Rafique and V. B. Miˇsi´c, “The effects of test-driven development on external quality and productivity: A meta-analysis,” IEEE Transactions on Software Engineering, vol. 39, no. 6, pp. 835–856, 2013.

[6] Davide Fucci, Burak Turhan, “On the role of tests in test- driven development: A differentiated and partial replication”, Empirical Software Engineering Journal (April 2014, Volume 19, Issue 2, pp 277-302)

[7] Tosun A., Dieste O., Fucci D., Vegas S., Turhan B., Erdogmus H., Santos A., Oivo M., Toro K., Jarvinen J., & Juristo N. An Industry Experiment on the Effects of Test-Driven Development on External Quality and Productivity

[8] Fucci, D., Turhan, B., & Oivo, M. The Impact of Process Conformance on the Effects of Test-driven Development (ESEM2014) 8th Empirical Software Engineering and Measurement, 2014 ACM/IEEE International Symposium on. Turin, Italy.

[9] Fucci, D., Turhan, B., & Oivo, M. On the Effects of Programming and Testing Skills on External Qualityand Productivity in a Test-driven Development Context (EASE2015) 19th Evaluation and Assessment in Software Engineering 2015 ACM/IEEE International Conference on., Nanjing, China.

[10] Viktor Farcic , Alex Garcia ; “Java Test-Driven Development”; First published: August 2015; Production reference: 1240815; Published by Packt Publishing Ltd.; Livery Place; 35 Livery Street; Birmingham B3 2PB, UK. ISBN 978-1-78398-742-9; www.packtpub.com; www.it-ebooks.in

[12] Christine_Sarikas (GENERAL_EDUCATION) https:// blog.prepscholar.com/independent-and-dependent-variables; Feb 12, 2018.

[12] https://chartio.com/learn/charts/what-is-a-scatter-plot/ Jan 9, 2019

[13] Svetlana Cheusheva; ttps://www.ablebits.com/ office-add ins-blog/2019/01/09/add-trendline-excel/ May 15, 2019.

[14] H. Munir, K. Wnuk, K. Petersen, and M. Moayyed, “An experimental evaluation of test driven development vs. test last development with industry professionals,” *Proc. 18th Int. Conf. Eval. Assess. Softw. Eng. - EASE ’14*, pp. 1–10, 2014.

[15] M. Moayyed, K. Petersen, H. Munir. Considering rigor and relevance when evaluating test driven dvelopment: A systematic review. Information and Software Technology, 2014.