SERVICES FOR SCIENCE AND EDUCATION
UNITED KINGDOM

TNC TRANSACTIONS ON NETWORKS AND COMMUNICATIONS

VOLUME 8, NO. 2
ISSN: 2054 -7420

Strength and Weakness of Regression Analysis and Quartile and IQR Deviation by using Test- Driven Development

[1]Myint Myint Moe, [2]Khine Khine Oo
[1,2]University of Computer Studies (Yangon), Myanmar;
myintmyintmoe@ucsy.edu.mm; khinekhineoo@ucsy.edu.mm

ABSTRACT

Test-Driven Development (TDD) derives an evolutionary approach which needs unit test cases to be made before implementation of the code. The goal of this paper is to examine product code quality and programmer productivity using test-driven development approach. This paper establishes regression analysis and quartile & inter-quartile range deviation to evaluate the process effect on external code quality and developer productivity. The results of this paper are that if developer productivity is the actual effect, external code quality will be fewer decreased and external code quality is the actual effect if developer productivity will be fewer reduced. Test-Driven Development can produce the clear of code, simple and bug-release. Defect numbers decreased when automated unit tests are written iteration related to test-driven development. If necessary, the code refactoring is acted TDD. This proposed system evaluates regression analysis and quartile & inter-quartile range deviation based on a fixed time-frame.

Keywords: Unit Test, TDD, Refactoring, External Quality, Developer Productivity.

1 Introduction

Unit testing verifies to do correct code and enhances the quality, reliability and the application software development cost. TDD begins with evolving and designing checks for every a small application function. In the standardized testing process, first create the code then test. If accept the test, the development group has to improve and refactor the code. Refactoring is a strength agile technique to improve existing software. Refactoring code refers altering some code without impacting its behavior. This approach assists to avoid duplication code as developers write small amount code at a time if accept tests. TDD approach decreases the defect density of developed software either immediately or in the long run and reduces debugging time. The objective of proposed system analyzes the effects of external quality and developer productivity on TDD. This system improves external code quality and speed up testing process or developer productivity. The aim of proposed system verifies the correctness of all codes and gives developers confidence by reducing code complexity that it is used actively over time and motivates developers to perform better code quality. The contribution of this paper observed external code quality (QLTY) and developer productivity (PROD) based on regression analysis and quartile and inter-quartile range deviation method.

This paper is organized as follows. Section (1) introduces about the Test-Driven Development. Section (2) describes a framework characterizing the whole procedure of test-driven development. Next, theoretical

DOI: 10.14738/tnc.82.8117
Publication Date: 30th April, 2020
URL: http://dx.doi.org/10.14738/tnc.82.8117

assessment of the proposed system is discussed in Section (3). Section (4) presents discussion and comparison of result. Finally, Section (5) concludes this paper.

2 Background Theory

Test- Driven Development is a coding approach that splits from the basic part of the agile code development approach. TDD speeds up the early development of tests, at the time changes are accepted and encouraged with functional components. Test-Driven Development invented by Kent Beck (inventor of Extreme Programming and JUnit) refers to a style of programming where three activities are Coding, Testing and Design. At first, its main idea is to perform initial unit tests for the code and then implement the actual feature of it. One of the features of software system requirement is user stories, are designed to simply describe and understand. These can be easily to exchange by the end user as they like during the project's handle time. The Test-Driven Development process is described in Figure 1. Step 1: Choose a user story; Step 2: Write a test that fulfills a small task of the user story and run this test. Then makes a failed test; Step 3: Rewrite the production code necessary to implement the feature; Step 4: Perform the pre-existing tests again, where any failed test exist. When the code is correct completely and finally goes to the refactoring stage; Step 5: As the refactoring stage is finished, the correct production code is made and the user can choose new user story again.

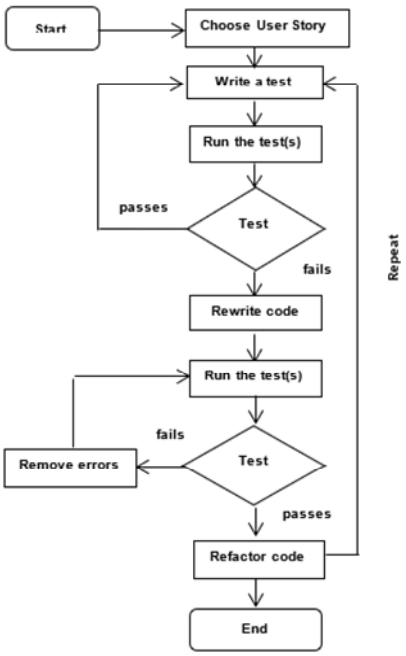


Figure: 1 Test-Driven Development process

3 Proposed System

In this proposed method, regression analysis and the quartile & IQR method applies to measure external code quality and developer productivity.

3.1 Research Questions

This system concentrates to analyze two results: external code quality and developer productivity.

RQ1 (RQ-QLTY): Does a higher number of tests illustrate higher external code quality?

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

The assumption of external code quality in RQ-QLTY and the assumption of developer productivity in RQ-PROD are focused on regression analysis and quartile & IQR deviation in descriptive statistics.

3.2 Method of Regression Analysis

This system collects fine-grained process data to examine the dependent variables. QLTY and PROD are the dependent variables. It applies regression analysis to express possible interactions such as external code quality and developer productivity. The development approach level: TDD will be applied as main factor. Table 1 summarizes the efficient rules applied by the IDE (Integrated Development Environment) plugin to suppose the type of development cycle according to the basic sequence of actions [13]. This system resolves the computation level of each process and the implications of the chosen metrics. The results under study are external code quality, expressed by a defect-based (or dually, correctness-based) metric (QLTY); developer productivity, expressed by an effort-based metric (PROD).

Type	Definition
Test-first	Test formation → Test accumulation error → Code correcting →Test failure→ Code editing→ Test pass
	Test formation → Code correcting → Test failure→ Code correcting → Test pass
	Test formation → Test accumulation error → Code correcting → Test pass
	Test formation → Code correcting → Test pass
Refactoring	Test correcting (file size changes ± 100 bytes) → Test pass
	Code correcting (number of methods, or statements decrease) → Test pass
	Test correcting AND Code correcting → Test pass
Test addition	Test formation → Test pass
	Test formation → Test failure → Test correcting→ Test pass
Production	Code correcting (number of methods unchanged, statements increase) → Test pass
	Code correcting (number of methods increase, statements increase) → Test pass
	Code correcting (size increases) → Test pass

3.2.1 External Code Quality

External code quality (*QLTY)* calculates how well the system matches the functional requirements. The external code quality (*QLTY*) performs the degree of agreement of the system with the functional requirements. The formula for computing *QLTY* is identified as [13]:

$$QLTY = \frac{\sum_{j=1}^{\#N} QLTY_j}{\#N} \times 100 \quad (1)$$

$QLTY_j$ = the quality of the j^{th} user story

#N = the total number of delivered user stories

where $QLTY_j$ *i*s the quality of the user stories j^{th} implemented by the subject. $QLTY_j$ is identified as:

$$QLTYj = \frac{\#ASSERT_j(PASS)}{\#ASSERT_j(TOTAL)} \quad (2)$$

$\#ASSERT_j(PASS)$ = passing assert statements from the associated test

$\#ASSERT_j(TOTAL)$ = total assert statements in the associated test

In turn, the number of user stories addressed (#N) is identified, such as

$$\#N = \sum_{j=0}^{n} \begin{cases} 1 & ASSERT_j(PASS) > 0 \\ 0 & otherwise \end{cases} \quad (3)$$

Where *n* is the number of user stories that compose the task. In both cases, it describes the number of passing JUnit assert statements in the set of tests associated with the j^{th} user stories. Consequently, a user story is considered addressed if it passes at least one of its JUnit assert statements. *QLTY* is a ratio calculate in the range [0,100].

For example, supposing that a person assesses eleven user stories (#*N* = 11), this means that there are eleven user stories for which at least one assert statement passes in the test suite. Let us assume that the acceptance test of the first analyzed user story contains 1 assertion, out of which one is passing. The acceptance tests of the second user story contain 11 assertions, of which seven are passing and so on.

i.e. ($QLTY2 = \frac{\#ASSERT_2(PASS)}{\#ASSERT_2(Total)} = \frac{7}{11}$ = 0.636)

i.e., ($QLTY = \frac{\sum_{j=1}^{\#N} QLTY_j}{\#N} \times 100 = \frac{1+0.636+0.5+0.571+0.5+1+0.4+0.5+0.375+0.428+1}{11} \times 100$ = 62.82%)

The table-2 dataset containing of QLTY and PROD attributes was tested to find deviations using MarsRover data-sets. This paper used the dataset from reference [15].

Table 2: To calculate the data-sets of MarsRove

Task	Passed Assert	Executed Assert
T1	1	1
T2	7	11
T3	4	8
T4	4	7
T5	4	8
T6	8	8
T7	6	15
T8	4	8
T9	3	8
T10	3	7
T11	8	8

Table 3: Solution of QLTY

	Assert pass	Assert total	QLTY
Q1	1	1	1
Q2	7	11	0.636
Q3	4	8	0.5
Q4	4	7	0.571
Q5	4	8	0.5
Q6	8	8	1
Q7	6	15	0.4
Q8	4	8	0.5
Q9	3	8	0.375
Q10	3	7	0.428
Q11	8	8	1
Total	52	89	62.82

3.2.2 Developer Productivity

Developer productivity (*PROD*) expresses the work done by the persons with the needed quality and within the identified time. Its formula is identified as [13]:

$$PROD = \frac{OUTPUT}{TIME} \quad (4)$$

OUTPUT describes the percentage of passing JUnit assert statements found in the set of tests for a task.

$$OUTPUT = \frac{\#ASSERT(PASS)}{\#ASSERT(TOTAl)} \times 100 \quad (5)$$

TIME (minutes) is an estimate of the amount of work used in the resolution of a task and is focused on the time records (milliseconds) collected by the IDE. If the person opened and closed the IDE several times during a run, *TIME* was computed as the sum of the all open-close intervals. The maximum time allocated to a task was ~300 minutes.

$$TIME = \frac{t_{close} - t_{open}}{6000} \quad (6)$$

For example, assume a person implements a task with a total of 89 assert statements in a test suite. After running the acceptance test suite against the person's solution, 52 assert statements are passing.

i.e. ($OUTPUT = \frac{\#ASSERT(PASS)}{\#ASSERT(TOTAL)} \times 100 = \frac{52}{89} \times 100 = 58.427\%$)

Suppose that the solution was expressed in one and a half hours (i.e., TIME = 90 minutes). The person's PROD is therefore 0.649, defining an assertion passing rate of 0.649 percent per minute.

i.e. ($PROD = \frac{OUTPUT}{TIME} = \frac{58.427}{90} = 0.649$ percent per minute)

3.2.3 Analysis

This system assumes that the descriptive statistics analysis of the information compiled from the code implementation by TDD responds positively to questions RQ1. A slight decrease of the productivity is assumed due to the fact that TDD acts more steps in its process RQ2.

Figure 2: External Code Quality

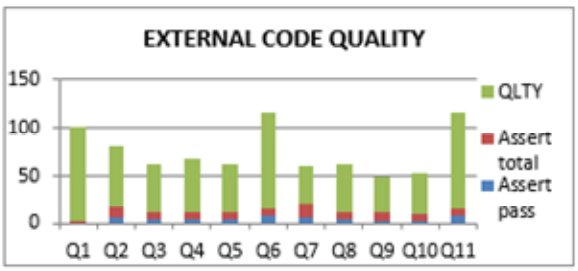
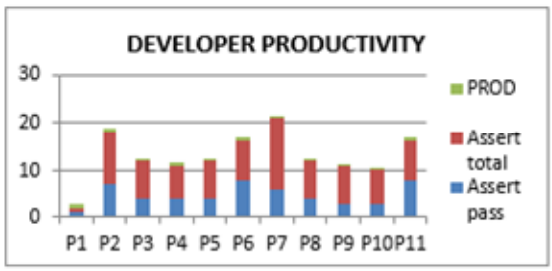
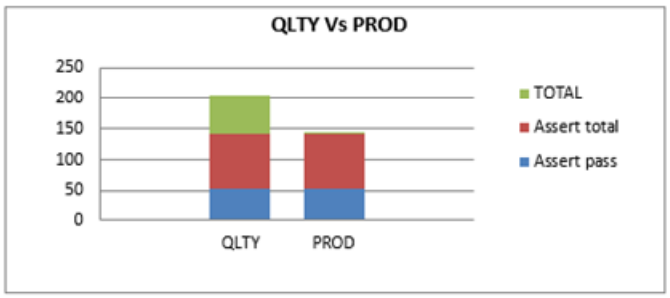


Figure 3: Developer Productivity



In figure 2: the external code quality is increased by calculating the defect-based of quality (QLTY). In figure 3, the developer productivity is slightly reduced by calculating the effort-based of productivity (PROD). In figure 4, the relationship of external code quality and developer productivity applied in this analysis appear higher external quality and the less developer productivity.

Figure 4: Interrelation of QLTY and PROD



3.3 Method of quartile and IQR deviation

The data set containing of QLTY and PROD attributes were examined to find derivations using both z-score and modified z-score methods [3,7,11] .Table 4 provides the raw data applied in the assessment. In the proposed system, quartile and IQR deviation are applied by examining to describe possible relationships such as external code quality and developer productivity. QLTY identified as the percentage of acceptance tests passed for the implemented stories. PROD identified as the percentage of implemented stories. Interquartile Range is most applied when comparing more than two data-sets. This statistical measure applies the concept of the median. The middle-ranking value in a range of data ordered from largest to smallest. It then discovers the median of the upper half (Upper Quartile) and subtracts the median of the lower half (Lower Quartile) to produce the difference between the quarter and three-quarters value are

called the Interquartile Range. The interquartile range is built on the calculation of other statistical measures. Before determining the interquartile range, first require to know the values of the first quartile and third quartile. (First and third quartiles depend upon the value of the median). This method requires to explore collected data from an area to compare with data from another area. The interquartile range is to subtract the first quartile from the third quartile. The table-4 dataset containing of QLTY and PROD attributes was tested to discover outliers using both z-score and modified z-score method**s** [3,7,11]**.** This paper applied the dataset from Davide Fucci and Burak Turhan [7].

Table-4 dataset consisting of QLTY and PROD attributes

ID	QLTY	PROD
1	69	100
2	28	46
3	49	92
4	72	100
5	78	92
6	75	100
7	60	69
8	69	85
9	26	46
10	43	31
11	68	100
12	86	100
13	68	85
14	11	46
15	75	54
16	55	85

Table 5: Data of computation for QLTY and PROD

	QLTY	PROD
MINIMUM	11	31
FIRST QUARTILE (Q1)	46	50
MEDIAN (Q2)	68	85
THIRD QUARTILE (Q3)	73.5	96
MAXIMUM	86	100
INTER QUARTILE RANGE (IQR)	27.5	46

For sample of QLTY,

Raw Data: 69, 28, 49, 72, 78, 75, 60, 69, 26, 43, 68, 86, 68, 11, 75, 55
Ascending Order: 11, 26, 28, 43, 49, 55, 60, 68, 68, 69, 69, 72, 75, 75, 78, 86

$\overleftarrow{11,26,28,43,49,55,60,68,}$ ↓ $\overrightarrow{68,69,69,72,75,75,78,86}$

Lower Half (Q1) Median Upper Half (Q3)

Minimum = MIN = Smallest value of all data set = 11
Median = If number (n) is even, average of two middle numbers of all data set
Median = If number (n) is odd, middle numbers of all data set
Median = $\frac{68+68}{2}$ = 68
Maximum = MAX = Largest value of all data set = 86

First Quartile Q1 = If number (n) is even, average of two the median of the lower half of the data set
= If number (n) is odd, the median of the lower half of the data set
= $\frac{43+49}{2}$ = 46
Third Quartile Q3= If number (n) is even, average of two the median of the upper half of the data set
= If number (n) is odd, the median of the upper half of the data set
= $\frac{72+75}{2}$ = 73.5
Inter Quartile Range = IQR = the difference between the first quartile and third quartile

= to subtract the first quartile from third quartile
= Q3 – Q1 =73.5 – 46 = 27.5

3.3.1 Assessment

The figure below is a box-plot**.** A box-plot is a graph that gives a good indication of how the values in the data are spread out. Box-plot is a systematized way for describing the distribution of data based on five numbers ("minimum", first quartile (Q1), median, third quartile (Q3), and "maximum"). Box-plot can tell about outliers that are applied when comparing distributions between many groups or data-sets. The figure below is a comparison of a box-plot of a nearly normal distribution and the probability density function for a normal distribution. This system expects that the descriptive statistics analysis of the information compiled from the developer productivity by TDD responds positively to questions *RQ2*. Decrease of the external code quality is expected due to the fact that TDD presents more steps in its process *RQ1*.

Figure 5: the external code quality is decreased by computing the IQR deviation of quality (QLTY). Figure 6: the developer productivity is improved by computing the IQR deviation of productivity (PROD). Figure 7: the relationship of external code quality and developer productivity applied in this assessment appear higher developer productivity and slightly lower external quality.

Figure 5: Quality

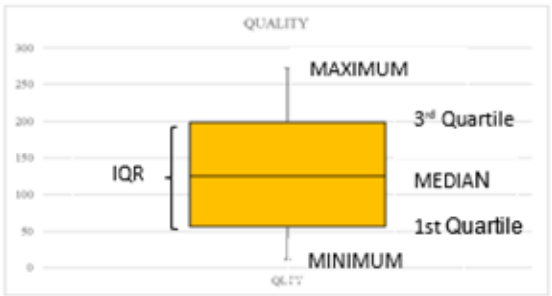


Figure 6: Productivity

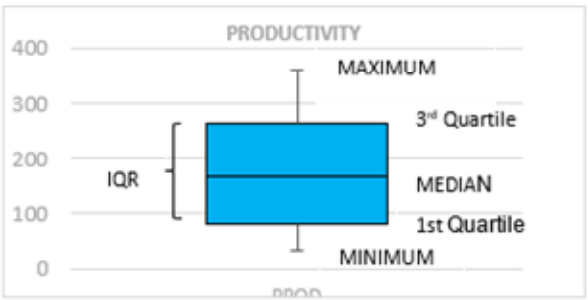
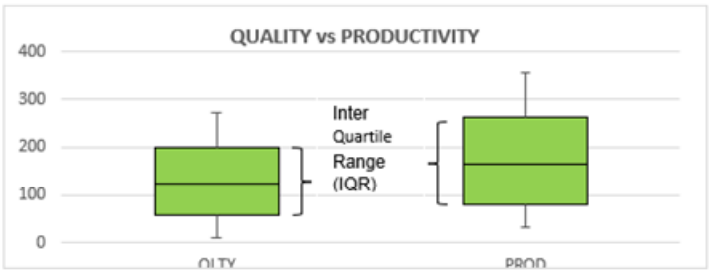


Figure: 7 Quality vs Productivity



4 Discussion And Comparison Of Result

In this section, this research paper discusses and compares the results of the regression analysis and quartile & IQR deviation. It discusses the current state of knowledge about the effects of external code quality and developer productivity according to regression analysis. Regarding the two research questions (RQ1 and RQ2), the external code quality (RQ1) is enhanced by computing the defect-based metric of quality (QLTY). The developer productivity (RQ2) is slightly decreased by computing the effort-based metric of productivity (PROD). Therefore, the relationship of external code quality and developer productivity applied in regression analysis appears higher external quality and deeper developer

productivity. In quartile & IQR deviation, regarding the two research questions (RQ1 and RQ2), the external code quality (RQ1) is lightly decreased by computing the passing assert statements from the acceptance test of quality (QLTY). The developer productivity (RQ2) is enhanced by computing the work done by the persons with the needed quality, within the identified time of productivity (PROD). Therefore, the relationship of external code quality and developer productivity applied in quartile & IQR deviation appear higher developer productivity and the less external quality. Consequently, External code quality becomes improvement, developer productivity becomes diminishment. If external code quality decreases, developer productivity increases.

5 Conclusion

In this paper, Test-Driven Development approach responds through unit testing which enhances the quality of the software and improves customer satisfaction. This approach is an effective tool to enhance developer productivity in long run. When this proposed system analyzes regression analysis, the outcome of external code quality is improved and the outcome of developer productivity is diminished. When this proposed system examines inter quartile range deviation, the outcome of external code quality is fewer decreased, the outcome of developer productivity is increased. The external code quality and developer productivity applied in this system evidently effect. If higher external code quality, lower developer productivity and vice versa, a developer could be more productive by implementing as much user stories as possible but ignoring external code quality.

REFERENCE

[1] K. Beck. *Test-driven Development*: Addison-Wesley, 2003

[2] 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).

[3] 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

[4] Y. Rafique and V. B. Mi˘si´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.

[5] Causevic, A., Shukla, R., & Punnekkat, S. *Industrial study on test driven development: Challenges and experience*, 1st International Workshop on Conducting Empirical Studies in Industry (CESI). 2013

[6] Davide Fucci; Burak turhan, *A Replicated Experiment on the Effectiveness of Test- First Development*; 2013 ACM/IEEE; International Symposium on Empirical Software Engineering and Measurement.

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

[8] 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*

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

[10] 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. Software. Eng. - EASE '14, pp. 1–10, 2014

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

[12] O. Dieste, E. R. Fonseca, G. Raura, and P. Rodríguez, *Effective Test-Driven Development: Un Experiment Replicado, Rev. Latinoam. Ing. Softw.*, vol. 3, no. 3, p. 141, 2015.

[13] Fucci D., Erdogmus H., Turhan B., Oivo M., & Juristo N; *A Dissection of Test-Driven Development: Does It Really Matter to Test-First or to Test-Last?* IEEE Transactions on Software Engineering (2016, in print) – Invited Journal First Paper (ICSE2017) Empirical Software Engineering Journal

[14] Tosun, A., Ahmed, Turhan, B., & Juristo; *On the effectiveness of unit tests in test-driven development*. Proceedings of the 2018 International Conference on Software and System Process - ICSSP '18

[15] 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/32*