ICIC Express Letters
Volume **15**, Number **6**, June **2021**

ICIC International ©2021 ISSN 1881-803X
pp. **545–552**

TEST FIRST VS TEST LAST: A STUDY OF SOFTWARE QUALITY IN ACTION

Wantana Sisomboon

Faculty of Informatics
Burapha University
169 Longhard Bangsaen Road, Saensuk, Muang, Chon Buri 20131, Thailand
wantanasi@buu.ac.th

Received October 2020; accepted January 2021

Abstract. *Scrum is an iterative and incremental approach to developing software. This approach breaks down large projects into small pieces of user stories that can be completed within time-boxed iterations. Each iteration is done in a process called a Sprint. A testing process is conducted at the end of each Sprint (Test Last). Test Driven Development (TDD) models strongly recommend increasing the quality of software with a short, fixed schedule of release cycles. TDD is an agile practice in which test cases are created before coding (Test First). This paper presents a comparative code quality and productivity case study of adopting Test Last and Test First for inexperienced programmers. Selected studies include experiments comparing Test First with control groups, then analysis of the test summary reports, online questionnaires, and focus groups. In addition, 13 experts, consisting of 3 lecturers and 10 experienced programmers, were assigned to evaluate software productivity and quality of the resulting software. This research describes results from experiments investigating the distinction between the Test First and the Test Last practices to determine. On average, the code quality produced by the two groups is slightly different. The Test First method allowed coders to produce better quality code. In addition, the Test First practice helped developers to complete test cases in approximately 91.02% of the total cases. Furthermore, the average time in development for all teams was decreased by 17.27%.*

Keywords: Test driven development, Scrum, Software development process, Inexperienced programmers, Code quality

1. **Introduction.** Testing is taken seriously by software development companies in Thailand. In software testing, it consists of 2 processes which are verification and validation. Verification is the process of checking whether the developer team built the software product correctly and the validation process is used to check that the product meets the user requirements [1]. The costs of software defects can be measured by the impact of the defects. The sooner defects are found, the less money is required to fix their cost [2].

Currently, companies apply various software development methodologies, depending on project resources and customer's requirements. Although most of the software companies quickly adopted the Agile technique as their process for producing software products, many defects were found in the User Acceptance Testing (UAT) process.

In contrast, many research efforts and experiments show that software products produced by Test Driven Development (TDD) practice gave better quality. TDD is an agile practice in which test cases are created before coding. However, developers themselves often do not select the TDD development process. This is due to fact that programmers assign a higher priority to coding than the testing process. Nevertheless, TDD or Test First models are strongly recommended for increasing the quality of software created using a short, fixed schedule of release cycles [3].

DOI: 10.24507/icicel.15.06.545

The main objective of this research study is to perform an experiment using undergraduate students in an academic software engineering environment as programmers and testers in order to investigate the distinction between the quality of code written using the Test Last and Test First practices and verify the assertions made previously.

2. **Literature Review.** This section is a review of works about adapting Scrum and test driven development approaches in software development. It reviews selected, relevant sources to provide a description, summary, and critical evaluation of these works in relation to the research problem being investigated.

2.1. **The background of Scrum.** Scrum is defined as a repetitive framework for software and product development. It is an Agile methodology which is one of the most famous project management methodologies. A key principle in Scrum is that during all phases of development, the customer is involved. The Scrum workflow consists of coordination between the Scrum team and the product owner. A cross-functional Scrum team includes developers, testers, and other specialists [4,5]. In this experiment, the Scrum model operated by dividing a project into small tasks is called Sprints. One task was writing a "User Story" [6] which allows non-technical people to define the high-level objective in a single sentence, structured as follows: "As a [persona], I [want to], [so that]". In the testing context, a user story generally includes a definition of "Done" which describes a completed task for a tester. Usually, testing in the Scrum Sprint is defined as Unit testing. The test is conducted at the end of a Sprint. In addition, a UAT test is conducted in the Sprint Review. In this paper this method is called "**Test Last**".

2.2. **Test Driven Development (TDD).** Test Driven Development (TDD) was introduced by Kent Beck in the late 1990s as part of the eXtreme Programming (XP) method. The bugs are found and fixed earlier when it is cheaper to fix them. This experiment adopted a TDD process which is defined by the 3 following rules: 1) Unless it is to allow a failed unit to test pass, developers are not permitted to write any production code; 2) Developers are not supposed to write more than adequate unit tests to fail, and compilation failures are failures; 3) Only necessary product code can be written by developers to make previously failing unit test pass [7,8]. The basic idea behind TDD is to write and fix failed tests before writing new code. In this paper this method is called "**Test First**".

2.3. **TDD family: BDD and ATDD.** This section aims to describe the different testing practices of TDD, ATDD, and BDD. Acceptance Test Driven Development (ATDD) is a software development methodology based on an agile framework where customers, developers, and testers are involved in the acceptance test. ATDD includes acceptance testing, but is done before writing code. It emphasizes writing acceptance tests [9]. Behavior-Driven Development (BDD) is known as a descendant of TDD and ATDD. BDD is an approach that defines the system's behavior as user stories that are executable. It reflects on the actions of the system for users who interact with the system and ensures that all participants in the project communicate in the same language. It simplifies the translation between the language of software development and the domain language of the user. Behavior-driven development allows project stakeholders to work together to ensure that the best software is created to meet the needs of the users and ensures that all participants in the project communicate in the same language [9].

2.4. **Refactoring.** The last process of TDD is refactoring. Refactoring is a transformation technique for existing code, improving its internal structure without altering its external behavior [10]. The idea of refactoring is to carry out the changes as a series of small steps without introducing new defects into the system. In this experiment, 13 refactoring techniques were selected from the research of Kaur and Singh in 2017. They

extracted the refactoring techniques from ten research papers [11] and identified the following types of refactoring: 1) Extract Method, 2) Extract Class, 3) Extract Super class, 4) Extract abstract class, 5) Move Method, 6) Code Transformations, 7) Replace Method with Method Object, 8) Replace Data Value with Object, 9) Encapsulate Field, 10) Replace Temp with Query, 11) Extract Aggregate Classes, 12) Wrap Return value, and 13) Safe Delete, and Replace Constructor with Builder method.

3. **Methods.** A formal experiment to assess internal consistency, programmer efficiency, and programmer expectations to equate test based creation with Scrum techniques was performed. 74 students were divided into 8 teams, but only 5 teams were chosen for this study. The other 3 teams changed their projects during the second cycle because the requirements from the sponsors changed, so the researcher decided to remove them from the experiment.

3.1. **Research questions.** The experiment was conducted to answer three research questions:

1) Which practice of testing will generate higher software code quality, Test First or Test Last?

2) What methods do developers use during refactoring to enhance the code?

3) What are the main advantages and disadvantages of using Test First and Test Last?

3.2. **Research design.** We carried out experimental tests with groups of 9 developers. The 45 students were divided into 5 teams. The developers were not familiar with TDD. They were trained in these methodologies during the project. The developers used the robot framework as an automatic test tool. The experiment was divided into 3 projects for 5 teams as shown in Table 1.

Table 1. Team project details

#	Method	Project 1	Project 2	Project 3	Project 4	Project 5
1	Project name	Online examination	Short course training	Recreation service management	Queue management	Case management
2	Product concept	Nursing 4.0	Nursing 4.0	Smart government	Smart government	Smart government
3	Number of test cases	67	150	53	62	341
4	Product size (LOC)	7,453	8,340	2,100	5,167	14,112
5	Testing level	Unit test, integration test	Unit test, integration test	Unit test	Unit test, integration test, system test	Unit test

The participants were sophomore year students. The purpose of the research was to prepare them for four important subjects for their future career. The main subject was team software development. The three other subjects were software testing, data analysis and visualization, and contemporary web development. All subjects were integrated into one project with a learning method called Project Based Learning (PBL). The students had background in basic web programming languages, but limited experience in complex system design and development. For most of the students, the average size of software they had worked on previously was only 7,000 LOC (Line of Code).

The duration of the experiments was 15 weeks. The software development process used was Scrum. The testing models were mixed, using both Test Last and Test First practices. The formal evaluation was divided into three cycles. At the end of each cycle, team developers met with customer representatives and team project assessors.

In Figure 1, Cycle 1 used the "Test Last" practices at the end of each Sprint, while Cycles 2 and 3 used the "Test First" practice. In the first cycle, software testing took place at the end of the Sprint. The duration of first cycle was 7 weeks, while cycles 2 and 3 lasted 4 weeks.

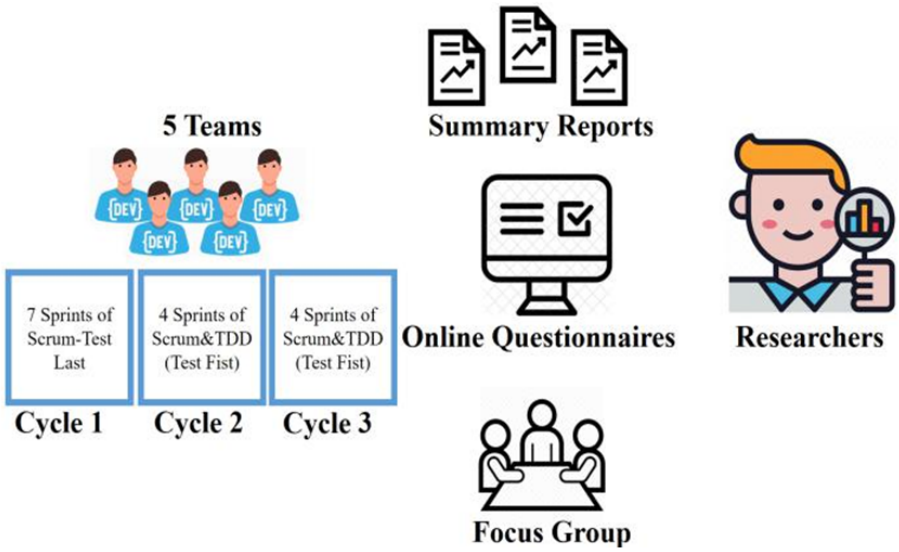

FIGURE 1. Research design

4. **Experiment Result.** In this section, the results of the quantitative and qualitative findings of the experiment are given. Both Test First and Test Last practices are essential for an efficient software development process. This study involved implementing projects that have a business priority to offer particular software products to consumers in close-to-industry environments, where measurement data is continuously collected.

4.1. **Quality of code.** In this experiment, the code coverage metrics were disabled in the Test First phase during this experiment. The decision to do that was based on the fact that Test First offers very high coverage and ensures that the entire functionality is well checked. In addition, for the accuracy of the code, in 2012, Cauevic and Team [12] presented the following formula to calculate the quality of code indicator:

$$C(x) = 1 - \frac{NF(x)}{N}$$

In this formula, x represents a specific participant of the experiment, N represents a total number of test cases produced by all participants, and $NF(x)$ represents the number of failed test cases [3]. Table 2 shows 1,149 test cases were produced by all participants, 667 test cases were produced for Test Last, while 476 were developed for Test First. Overall,

TABLE 2. Comparison of Test First and Test Last test cases

Team#	**LOC**	**Test Last**				**Test First**			
		Total	**Pass**	**Fail**	$C(x)$	**Total**	**Pass**	**Fail**	$C(x)$
1	7,453	67	60	7	0.99	67	65	2	0.97
2	8,340	150	132	18	0.97	66	66	0	1.00
3	2,100	47	40	7	0.99	53	40	13	0.97
4	5,167	62	62	0	1.00	62	62	0	1.00
5	14,112	341	251	90	0.87	228	186	42	0.91
Total	**37,172**	**667**	**545**	**122**	**0.82**	**476**	**419**	**57**	**0.88**

964 test cases passed and 179 failed. Code quality of the Test First group was slightly higher than that of the Test Last group. Only Team 3 using Test First produced lower code quality with Test First. In contrast, overall results show that quality of code from Test First was slightly different from Test Last. In the aspect of error code, Test Last produced 18.13% of code defects while Test First produced fewer errors at 11.97%. Test First was slightly better. They had 1 month more experience with Test Last than Test First. In our experiments, Test First produced slightly higher quality result than Test Last.

4.2. **Testing results from Scrum and TDD methods.**

4.2.1. *Development time analysis.* Table 3 illustrates a comparison between Test First and Test Last practices. The table shows data related to a percentage of decreased development time using Test First compared with using Test Last. The results show that every team spent less time developing projects when using Test First. The reduction in time of teams 1, 2, and 3 was more than 20 percent, while teams 4 and 5 spent less than 10 percent less time. In contrast the number of test cases for Test First of teams 4 and 5 is four times larger than for their Test Last. In conclusion, Test First reduces time spent in development better than using Test Last.

Table 3. Comparison of development time between Test First and Test Last practices

Team#	**Test Last development time (Hours)**	**Test First development time (Hours)**	**Time reduction**
1	150	111	26.00%
2	120	88	26.67%
3	100	78	22.00%
4	115	108	6.09%
5	125	118	5.60%
Total	**610**	**503**	**86.36%**

4.2.2. *Refactoring analysis.* Information about refactoring was collected manually, and then recorded in a testing summary report. The report shows the results with fail and pass codes export from an automation tool called robot framework. A walkthrough technique was adopted to measure the code quality. This technique is an informal review led by an assessor. Assessors are programmers with 5 to 10 years of experience. A refactoring checklist was designed and evaluated by specialists, and then adopted for this experiment. In addition, ten months after the experiment more data was collected from the same group of participants to evaluate and monitor the behavior of developers. The participants were educated at real software companies as student trainees for 16 weeks. The evaluations were made by the author using information collected via online surveys and interviews.

There are 10 different signs that code needs refactoring, as shown in Table 4. The percentages of participants in the "Refactoring needs" column indicate that "Lack of comments" happened frequently. It is a common mistake made by many junior developers when writing software. The next most frequent refactor needs are "Redundant code" and "Long methods, functions, or parameter lists". In contrast, the second evaluation found that "Lack of comments" decreased rapidly while other refactoring needs were increasing.

Table 5 shows refactoring activities as mentioned in Section 2. 7 activities were adopted for the project development in TDD model as shown in Table 2. The most popular activity was "Extracting the method makes the code simpler". The next most popular activities were 1) Renaming variables, files, classes and functions, 2) Adding comments

Table 4. Refactoring analysis

#	Refactoring needs	1st evaluation	2nd evaluation	Remark
1	Redundant code	22.5%	29.41%	Increase
2	Lack of comments	**30%**	**11.76%**	**Decrease**
3	Long methods, functions, or parameter lists	22.5%	23.53%	Increase
4	Difficult to understand	15%	17.65%	Increase
5	Contains difficult code	15%	17.65%	Increase
6	Conditional complexity	15%	N/A	N/A
7	Inconsistent names	7.5%	N/A	N/A
8	Many global variables	7.5%	N/A	N/A
9	Temporary fields	7.5%	N/A	N/A
10	Large argument list in procedure calls	7.5%	N/A	N/A

Table 5. Refactoring activities analysis

#	Refactoring activities	Adopting teams	Frequency
1	Renaming variables, files, classes and functions	1, 2, 3, 4	4
2	Extracting the method	All	7
3	Adding comments in controllers, model, helpers and views	1, 2, 3, 4	4
4	Substituting algorithms	1, 5	2
5	Breaking complex functions into smaller parts	1, 2, 3, 5	4
6	Extracting classes	1, 3	2
7	Removing a variable that is acting as a control flag for a series of Boolean statement	2	1

in controllers, model, helpers and views, and 3) Breaking complex functions into smaller parts.

5. **Discussion.** To investigate the distinction between the Test First and Test Last approaches, the feedback from 45 students in our software testing course was evaluated. In this section, the research questions from Section 3.1 will be answered.

5.1. **Question 1: Which practice of testing will generate higher software code quality, Test First or Test Last?** The results in the tables from previous chapter [Tables 1, 2, 3, 4 and 5] show that Test First code quality and productivity are slightly better than Test Last.

The code quality produced by the two groups was slightly different. Test First resulted in slightly better quality code (6.16 percent higher). In addition, Test First helped developers to complete test cases in approximately 91.02 percent of the total cases. Furthermore, the average time in development for all teams decreased by 17.27%.

5.2. **Question 2: What methods do developers use during refactoring to enhance the code?** The participants indicated that the following were the top 4 refactoring methods used during the project.

1) The code was made easier to understand by all teams using "Extracting the method".

2) Four teams used 1) "Renaming variables enhances readabilit", 2) "Adding feedback and renaming variables enhances maintenanc", and 3) "Breaking complex functions into smaller parts makes it much easier to change an algorithm".

3) Two teams used 1) "Substituting algorithms provides benefits for efficiency" and 2) "Extracting classes simplifies the code".

4) One team used "The program stays clean by extracting groups".

5.3. **Question 3: What are the main advantages and disadvantages of using Test First and Test Last?**

Test Last Advantage: Students are more familiar with Test Last than Test First. In addition, during the cooperative education course, most of the development teams in software companies were using Test Last. Some students state that Test Last provided ideas for creating more test cases because they had more experience with the system they were assigned to test. **Test Last Disadvantage:** Because of the flexible or unclear requirements, sometimes testers created unnecessary tests. Furthermore, many software project teams cannot test their project because the coding phase lasts longer than the Sprint time box.

Test First Advantage: Test First is better because they can plan test cases in advance and the test cases were written according to the system requirements. In addition, refactoring helped to improve code and software quality. It helps developers to write test cases to cover all important functions before they start developing software. As a result of the refactoring process, their code was easy to understand and readable. Test First helped to reduce adding new features during development because the code was arranged properly, and in the refactoring phase, a team can improve the code structure. **Test First Disadvantage:** From the observations by a focus group, the researchers found that new testers and developers cannot adopt Test First until they have knowledge about automated testing and refactoring.

6. **Conclusion and Future Work.**

This study presents the outcome of an experiment in the software engineering field with undergraduate students which was performed to explore the difference between the Test First and the Test Last practices. On average, the code quality produced by the two groups were slightly different. The Test First method allowed coders to produce better quality code. In addition, the Test First practice helped developers to complete test cases in approximately 91.02% of the total cases. The average time spent in development for all teams was decreased by 17.27%. Test First is better than Test Last because developers could plan test cases in advance and the test cases were written according to the system requirements. In addition, refactoring helped to improve code and software quality. On the other hand, students are more familiar with Test Last than Test First and new testers and developers cannot adopt Test First until they have knowledge about automated testing and refactoring. However, this study has a few limitations. First, only one reviewer manually performed the qualitative study of the Internet questions in the survey. Therefore, the review may be biased. Although the actions of 45 students were evaluated in terms of generalizability, our study is not comprehensive. Second, during the experiment, participants reported their defects at the end of a cycle, and then the defects were corrected. In future work this should be reconsidered and the defects collection tool re-designed. Third, the refactoring methods were reported as group work. The result of the experiment would be more reliable if we collected the data individually. Finally, it might be better to study and compare "the software testing techniques/methodologies/frameworks" instead of comparing an agile framework with a direct testing driven development. In addition, it is important to consider establishing a proper security management mechanism in which each tester, as well as the developer, can protect their work in different stages of software development, including databases, servers, resources, and so on, in order to safeguard all the functions of the software development process.

Acknowledgment. This work is partially supported by Faculty of Informatics, Burapha University.

REFERENCES

[1] Y. Ghadi, M. Sh. Daoud, F. Kharbat and T. Elamsy, Evaluation of the difference between verification and validation of software and analyzing the significance among both, *ICIC Express Letters, Part B: Applications*, vol.10, no.10, pp.885-893, 2019.

[2] B. George and L. A. Williams, An initial investigation of test-driven development in industry, *Proc. of the 2003 ACM Symposium on Applied Computing (SAC)*, Melbourne, FL, USA, pp.1135-1139, 2003.

[3] A. Nanthaamornphong, Test-driven development in HPC science: A case study, *The Computing in Science & Engineering*, pp.98-113, 2018.

[4] A. Srivastava, S. Bhardwaj and S. Saraswat, SCRUM model for agile methodology, *2017 International Conference on Computing, Communication and Automation (ICCCA)*, Greater Noida, pp.864-869, 2017.

[5] F. Hayat, A. U. Rehman, K. S. Arif, K. Wahab and M. Abbas, The influence of agile methodology (Scrum) on software project management, *The 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Toyama, Japan, pp.145-149, 2019.

[6] F. Dalpiaz and S. Brinkkemper, Agile requirements engineering with user stories, *IEEE the 26th International Requirements Engineering Conference (RE)*, Banff, AB, Canada, pp.506-507, 2018.

[7] Z. Khanam and M. N. Ahsan, Evaluating the effectiveness of test-driven development: Advantages and pitfalls, *International Journal of Applied Engineering Research*, vol.12, no.18, pp.7705-7716, 2017.

[8] N. Agarwal and P. Deep, Obtaining better software product by using test first programming technique, *The 5th International Conference – Confluence the Next Generation Information Technology Summit (Confluence)*, Noida, India, pp.742-747, 2014.

[9] M. Alhaj, G. Arbez and L. Peyton, Approach of integrating Behaviour-Driven Development with Hardware/Software codesign, *International Journal of Innovative Computing, Information and Control*, vol.15, no.3, pp.1177-1191, 2019.

[10] I. Verebi, A model-based approach to software refactoring, *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Bremen, pp.606-609, 2015.

[11] G. Kaur and B. Singh, Improving the quality of software by refactoring, *International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, pp.185-191, 2017.

[12] A. Cauevic, S. Punnekkat and D. Sundmark, Quality of testing in test driven development, *The 8th International Conference on the Quality of Information and Communications Technology*, Lisbon, Portugal, pp.266-271, 2012.