Bulletin of Electrical Engineering and Informatics
Vol. 7, No. 1, March 2018, pp. 80~89
ISSN: 2302-9285, DOI: 10.11591/eei.v7i1.897

Test First Model for Application in the Academic Setting

Normi Sham Awang Abu Bakar[1], Norzariyah Yahya[2]*
[1]Department of Computer Science, International Islamic University, Malaysia
[2]Centre for Foundation Studies, International Islamic University, Malaysia

Article Info

Article history:

Received Nov 26, 2017
Revised Jan 27, 2018
Accepted Feb 10, 2018

Keywords:

Pilot experiment
Test first
Test last

ABSTRACT

This research elaborates the selection of the Test First and Test Last model for a pilot experiment that was executed as a feasibility study to validate the suitability of the existing Test First model for its implementation in the series of actual experiment. The series of actual experiment is designed to investigate the quality of the project developed by the students in higher educational institution with the Test First over Test Last model. The findings gathered from the pilot experiment demonstrate that there were misunderstandings on the user stories among the participants that have led to the development of an enhanced Test First model.

Corresponding Author:

Norzariyah Yahya,
Centre for Foundation Studies,
International Islamic University, Malaysia.
Email: norzariyah@iium.edu.my

1. INTRODUCTION

Test First is also known as Test Driven Development [1]. Test First is a process that requires a developer to design test cases that originate from a set of requirements, and the process continues with the development of a production code, which is written to pass the designed test cases. All coding processes are implemented in a small chunk of tasks, iteratively and incrementally. The basis of Test First implementation consists of three iterative phases [2]; Red (writing a unit test, and the unit test will automatically fail), Green (writing production code and the actual code is then tested by the unit test, and it must pass the test cases), and Blue (refactoring the code).

Madeyski [3] promoted Test First as one of a tangible practices in eXtreme Programming (XP), a part of Test First as declared in Agile manifesto. In their work, Desai, et al. [4] stated that Test First tends to aid students with the design of complex projects and increases the students' confidence. In addition, Janzen and Saiedian [5] reported that Test First programmers will be more likely to write software in more and smaller units, and the software will be less complex and more highly tested. While, Janzen and Saiedian [5] that Test First programmers might write more highly coupled program in smaller units. Additionally, it has possibility to increase in abstractness which indicates that the higher coupling is a good coupling that resulting in more flexible software. Nevertheless, Janzen and Saiedian [5] are unable to substantiate claims that Test First improves cohesion.

Moreover, studies on the Test First implementation in universities have reported quality of software by measuring the internal and external quality [6]-[8]. However, results from these works demonstrated various outcomes of Test First implementation. Pančur and Ciglarič [6] reported that similar quality products were produced by the students regardless of whether by using Test First, or by executing a more traditional testing approach. In other research, both Huang and Holcombe [7] and Stevens, et al. [9] stated improvement in productivity by implementing Test First, although Huang and Holcombe [7] result from the latter was not statistically significant. Meanwhile, Vu, et al. [8] found Test Last produced 39% more defects compared to

Test First. Additionally, Causevic, et al. [10] investigated the differences between the quality of test cases created using Test First and Test Last model. They found that mutation quality and falling assert statements for both models are similar. In a study by Kaufmann and Janzen [11] it was reported that the quality of the projects was determined by using the number of passed JUnit tests, and subjective score was assigned by the instructor based on the code coverage. For clarity, all the above researchers adopted JUnit in their experiment in measuring the quality of the projects involved in their studies. In short, based on the comments received from the students involved in the experiments, those who prefer Test First implementation claimed that the testing process helps in providing the developers with confidence [12], [13], the ability to produce higher quality result [7, 14], simpler design [12] and fewer defects [12].

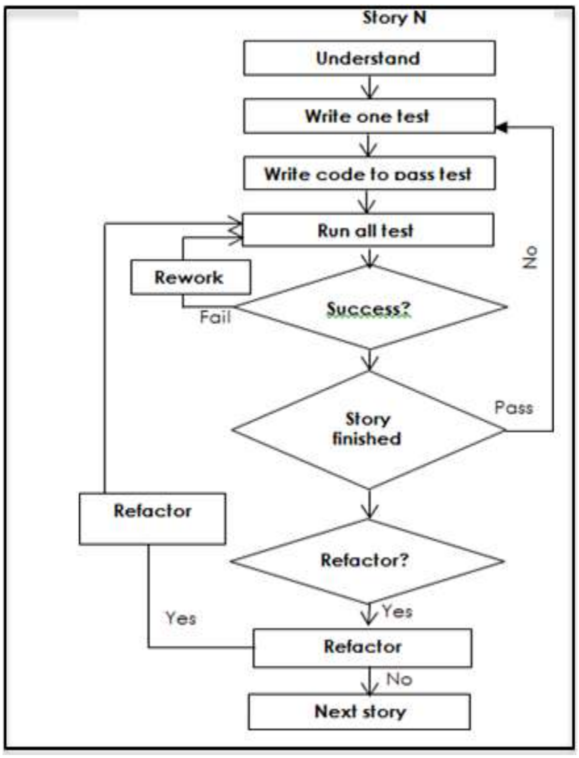
Therefore, the introduction of Test First in educational settings is important as a platform to impart to the students the basics of Agile principles. In addition, Causevic, et al. [14] stated that training, test cases design and knowledge on the Test First are the factors needed to ensure the success of Test First implementation in industrial settings. Hence, training and knowledge should be imparted to the students before the adoption of the Test First techniques. Moreover, the mixture of results in the Test First implementation as reported in other research, has motivated us to conduct study on their suitability for implementation in our educational settings.

1.1. The Existing Test First/Test Driven Development Model

Series of experimental studies comparing Test First with other traditional methods have been conducted for more than a decade. Such as the experiments implemented by Buffardi and Edwards [15], Funabiki, et al. [16] and Lappalainen [17] impart tools such as on-line automated grading tool to students, and provide the with feedbacks that are known as Web-CAT, Web-based Java Programming Learning Assistant System (JPLAS), and ComTest, all of which support Test First implementation. Additionally, Janzen and Saiedian [5], Causevic, et al. [10], and Lemos, et al. [18] have also conducted series of experiments to compare the quality of code produced using Test First model with those produced using a more traditional method. Nevertheless, from all of the above-mentioned studies, only research carried out by Pančur and Ciglarič [6], Yenduri and Perkins [19], Madeyski [20] and Janzen and Saiedian [5] clearly explained in detail about the models that they introduced in their research.

Specifically, Pančur and Ciglarič [6] conducted a research comparing Test Driven Development, Iterative Test Last and Test Last with Coarse Granularity. Their research objective is to find out if there is any effect of Test Driven Development on Agile development process, the resulting code and tests. They measured the productivity by the number of implemented user stories per hour (NIUSPH), percentage of acceptance tests passed (PATP), code complexity, test thoroughness (branch coverage) and fault-finding capabilities of tests which is measured as mutation score indicator (MSI). The result of the experiment shows that there are no statistically significant differences between Test Driven Development and Iterative Test Last regarding productivity, code complexity, branch coverage, percentage of acceptance tests passed and mutation score indicator. Therefore, they concluded that the benefits of Test Driven Development compared to Iterative Test Last Development are small and thus in practice relatively not important, although the effects are positive. There is an indication of Test Driven Development endorsing better branch coverage, but the effect size is considered small. The model used by Pancur and Ciglaric is illustrated in Figure 1.

Yenduri and Perkins [19] conducted an experiment involving senior undergraduate students who were instructed to develop a software performing unit testing in the conventional way. As illustrated in Figure 2, developers have to develop the test cases after the development phase and extract the test cases before implementation as in Agile Programming incrementally and iteratively for both Test Driven Development and Iterative Test Last approaches. The results showed with Agile programming, less number of faults in the software were found. In addition, the quality of software was better and the productivity increased. Their result showed the Test Driven Development approach produced better quality and productivity. This is due to the way the test cases were designed as the software was being developed. Their experiment focuses on smaller subsets of information and a large number of test cases designed along with program development lessened the rework time and promoted quality. Their experiment results encourage the employment of Test Driven Development approach but the implementer should consider the size of the software project being implemented. Yenduri and Perkins [19] were confident that Test Driven Development approach may be a good choice in comparison with the traditional approach for small sized software projects. However; they proposed that the usefulness of the approach needs to be validated by using it for larger projects involving larger group numbers.



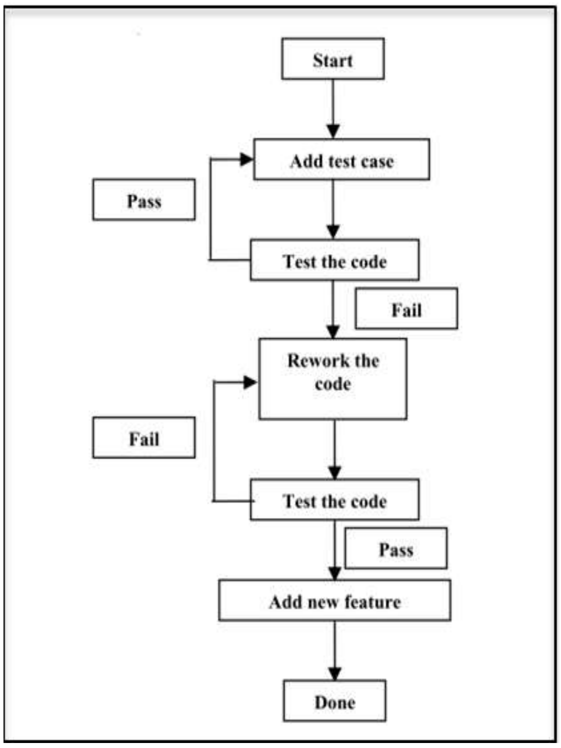


Figure 1. Pancur & Ciglaric's test driven development model

Figure 2. Yenduri and Perkin's test-driven development model

Whereas, Madeyski [20] model as shown in Figure 3 runs an experiment with the involvement of MSc students as subjects in the development of a web-based paper submission and review system in Java. She investigated the thoroughness and the fault detection effectiveness of unit tests, measured by branch coverage and mutation score indicator. The purpose of her research is to evaluate the impact of the Test First technique on unit tests. The subjects were randomly assigned to Test First and Test Last groups. She performed multivariate analysis in order to further reduce pre-existing differences among subjects, and to get a more sensitive measure of her experimental effect. The results indicated that there is no statistically significant difference between Test First and Test Last practices on the combined dependent variables (branch coverage and mutation score indicator), even if control is taken for the pre-test results, the subjects' experience, and when the subjects who showed deviations from the assigned programming technique are excluded from the analysis. Therefore, she concluded that the preliminary results presented in her paper show that, the benefits of the Test First practice in this specific context can be considered minor. However, the limitation is there is a probability that the first- ever experimental evaluation of the impact of Test First programming on mutation score indicator of unit tests is imprecise and further experimentation is needed to establish evidence. In addition, The similar model is executed in Madeyski and Szała [21] and the findings shows that the productivity of the codes developed by the students is higher and the quality of the codes is improved.

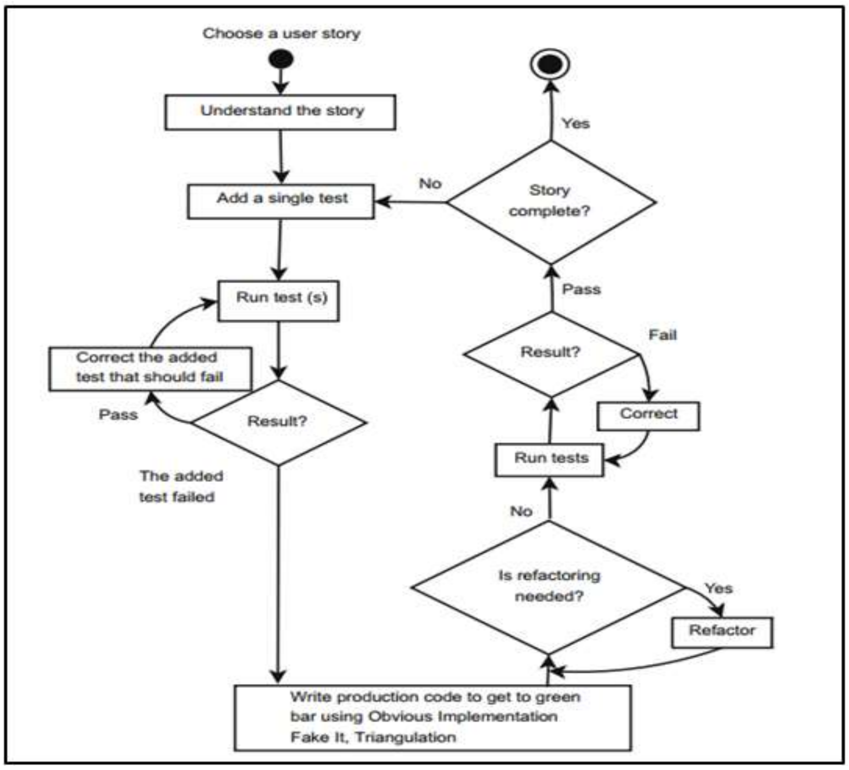


Figure 3. Madeyski's test first model

1.2. Model for Pilot Experiment

Four models introduced by Pančur and Ciglarič [6], Madeyski [20], Janzen and Saiedian [5] and Yenduri and Perkins [20] were shortlisted for the Test First model to be implemented in the Pilot Experiment and the comparison on the models is summarized in Table 1. Of those models however, only models applied by Yenduri and Perkins [19] and Janzen and Saiedian [5] have reported positive outcomes in using Test First. The Test First model (as illustrated in Figure 4) which is introduced by Janzen and Saiedian [5] uses detailed design as part of the development process and has been adopted in industrial settings.

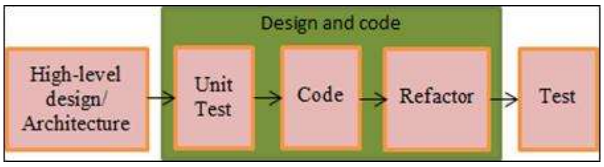


Figure 4. Janzen and Saedian's test first model

Table 1. Comparison of Existing Test First Models

Author	Janzen and Saiedian [5]	Pančur and Ciglarič [6]	Yenduri and Perkins [19]	Madeyski [21]
Participants	Professionals	Students	Students	Students
Outcome towards Test First	Positive	Negative	Positive	Negative [a]
Using User Stories	X	√	√	√
High Level Design	√	X	X	X
Test First Iterative phases defines by Fucci, et al. [2]	√	√	X	√

([a] the similar model is used in Madeyski and Szała [22] with the result that positive towards Test First.)

Additionally, a study by Humphrey [21] also indicated that detailed design is a part of the software development process, and it is a necessary development process mainly for the industrial settings. Whereas, the Test First model introduced by Yenduri and Perkins [19] does not contain a refactor component which is one of the elements in Test First processes. Therefore, Yenduri and Perkin's model was not selected for the pilot experiment while Janzen and Saiedian's model is also avoided since it is the method, which is adopted in industry.

Model in Figure 4 and model introduced by Janzen and Saiedian [5] show the Test First models which contain the unit test and refactor in the diagram flow. Thus, it makes these models suitable as a Test First model to be used in the pilot experiment. Additionally, the outcome for Pančur and Ciglarič [6] denotes insignificancy in the productivity, code complexity, branch coverage, percentage of acceptance tests passed and mutation score indicator for the Test First model. However, this pilot experiment adopted the model by Pančur and Ciglarič [6] despite all the insignificancies in its outcome. This is due to the Test Last model in Madeyski [21] guided the students to develop the test cases at the beginning of the development which is contradictory to Pančur and Ciglarič [6], Yenduri and Perkins [19] and Janzen and Saiedian [5] while Janzen and Yenduri's model is used by practitioners. Therefore, the model developed by Pančur and Ciglarič [6] is adopted in the Pilot Experiment.

2. RESEARCH METHOD

The objective of this pilot experiment is to validate that the existing Test First model is suitable for the implementation in educational settings. In the pilot experiment, students were randomly assigned to either a Test First model or Test Last model. Moreover, series of classes on Test First and Test Last were also conducted for these students to teach them how to write unit test cases. The students were also exposed to the Pancur and Ciglaric's model which was used as the model in the pilot experiment. The students were given sets of user stories and they were guided on the development process depending on the model that was assigned to them. Their tasks was to develop a web application program with unit test cases implementation based on the Test First or Test Last model and the students were allowed to have a group member of two and maximum three members per group.

Essentially, designing the experiment was one of the crucial tasks. Particularly, the phases of experiment was developed based on Juristo and Moreno [22] as shown in Figure 5 experiment design started with the definition of objectives which was used as a guide to develop hypotheses. The hypotheses aid in designing the experiment design started with the definition of objectives which was used as a guide to develop hypotheses. The hypotheses aid in designing the experiment that determined the inquiry process, the research models to adopt, the experiment flow, the treatments, the sample of population, the number of replications and the instruments. The experimental design is executed in a manner to answer the research questions and to test the hypotheses. The final step is identifying the expected outcome and the method to analyse.

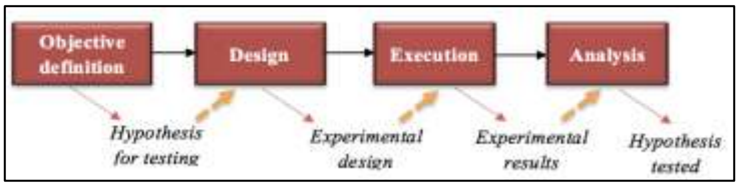
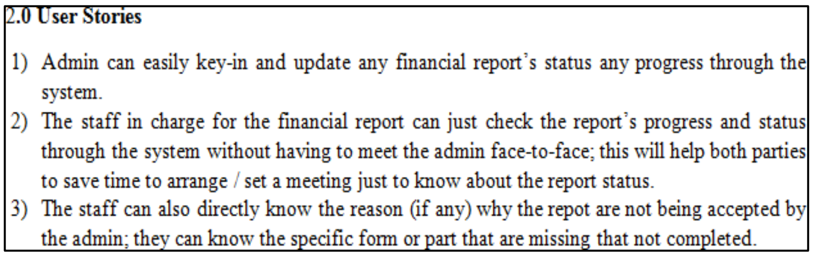


Figure 5. Experiment design based on Juristo and Mureno

The experiments are started by randomly assigning the students with either *Test First* or *Test Last* model. The randomly assigned model is executed by letting the students handpick a piece of paper (the paper was written with either *Test First* or *Test Last*). The students who picked *Test First* will have to implement *Test First* and students who picked *Test Last* will have to do *Test Last.* All students were given a series of JUnit trainings to teach them the annotations in JUnit and how to write test cases. The hands-on exercises should enable them to understand and to develop test cases. The students were also introduced to either the *Test First* or *Test Last* model. The attendance was made compulsory to all students. Crucially, the students involved were not introduced to both models. Instead, they were only aware of one model, either *Test First* or *Test Last*.

3. RESULTS AND ANALYSIS

There were fifteen projects submitted, however, only four groups managed to develop test cases while eleven groups failed to do any test cases. From their project report and presentation, we identified that the students did not understand the role of user stories. Although the students were informed that they have to develop their project based on the user stories statements, however, they replace and rephrase the user stories with statements that were mapped to their project outcome. Moreover, the students claimed that they cannot identify which object need test cases based on the given user story. The example of user stories that is recreated by the students based on the project that they developed is as shown in Figure 6.


2.0 User Stories

1) Admin can easily key-in and update any financial report's status any progress through the system.
2) The staff in charge for the financial report can just check the report's progress and status through the system without having to meet the admin face-to-face; this will help both parties to save time to arrange / set a meeting just to know about the report status.
3) The staff can also directly know the reason (if any) why the repot are not being accepted by the admin; they can know the specific form or part that are missing that not completed.

Figure 6. User stories given to students

In addition, the students also had no knowledge of appropriate class or method to use for writing test cases based on the given user stories. As the user stories concept was totally new to the students, they struggled to understand the role of user stories while at the same time they were trying to develop the test cases. Therefore, in ensuring the effectiveness of Test First implementation in educational settings, the students should be steered to focus in the Test First or Test Last without incorporating any new approach such as user stories. Despite being adopted in the pilot experiment, the Test First model introduced by Pančur and Ciglarič [6] will not be applied for the series of actual experiments. Test First model in Figure 6 has been proposed to be aligned with the findings in the pilot experiment, which does not favour the use of user stories. The usage of user stories should be avoided since user stories are totally new for the students and they struggled to understand the importance of user stories while at the same time they have to develop unit tests.

In detail, in the pilot experiment, the implementation of Pančur and Ciglarič [6] model has contributed to misunderstanding among students on the role of user stories beside their task to develop the test cases. Moreover, the students who refer to the given user stories claimed they were not able to identify the class or method needed in the test cases and the students even rewrite the user stories in order to fit in their project outcomes. Likewise, Causevic, et al. [14] also stated that insufficiency of design is a hindrance for developers to implement Test First. Recently, research conducted by Rumpe [23] proposed Unified Modeling Language (UML) as the tools to assist in Agile development, in fact Rumpe [23] suggested modeling as primary artifact for requirements and design documentation, code generation and test case development. Therefore, a Test First Model in Figure 6 is proposed. The model has been developed based on the students' existing knowledge and experience in system design, which is a combination of Test First developed by Pančur and Ciglarič [6], and by Janzen and Saiedian [5].

Janzen and Saiedian [5] is selected as the model to be combined with Pančur and Ciglarič [6] due to its positive result in producing less complex and highly tested project. In particular, the difference between Janzen and Saiedian [5] and Pančur and Ciglarič [6], Yenduri and Perkins [19] and Madeyski [20] is, Janzen did not apply the user stories as part of the Test First development flows. Moreover, the systematic review by [14] identified one of the factors that limits the adoption of Test First in the insufficiency of design which refers to the activity of structuring (or re-structuring) the system or software under development or in evolution in order to avoid architectural problems and to improve architectural quality. Therefore, implementing system design helps the students on the architectural part of their system development, thus, their project will be well designed to act as a guidance in the development phase. Additionally, Stevens, et al. [9] found that good software development methods begin with flow charts and diagrams. Recently, Latorre [24] reported the increase of the quality of projects developed using the Test First approach and specifically mentioned Unit Test Driven Development (UTDD). Interestingly, Latorre [24] also used high level design and architecture in the experiments. High level design is a form of architecture that described the projects the students intended to develop. It consists of a high-level model that reflects the current understanding and the future state architecture of the project, and one of the architecture blueprints is the UML artifacts Pressman [25]. The students involved in this pilot experiment are familiar with UML as they were taught UML in their first year of study. Therefore, the proposed model shown in Figure 6 is based on Janzen and Saiedian [5] which is using case diagram, class diagram and sequence diagram in the design stage with Pancur and

Ciglaric's Test First model with writing test cases, production code and refactor processes proposed for the future experiments.

3.1. Proposed Test First Model

As illustrated in Figure 7 the Test First model for students' implementation starts with requirements gathering that necessitates the students to propose their project title and briefly describe the project based on the stipulated criteria. Their proposed title and project descriptions will be evaluated, and project that fulfills the required criteria will then be allowed to proceed with detailed design stage. The students will continue to develop the detailed design comprising use case, class, and sequence diagrams. Next, the students will write the test classes and continue with the coding, iteratively. All students who will be involved in this experiment are already familiar with the project design using UML and they were exposed to the software lifecycle process in System Analysis and Design courses. The students are taught to design their project using diagrams in the above-mentioned courses. Furthermore, the Agile concepts such as user stories is new to the students and the students in this experiment will only be introduced to the Agile approach in Software Engineering course either in their third year or fourth year of studies. Figure 8 illustrated the proposed test last model.

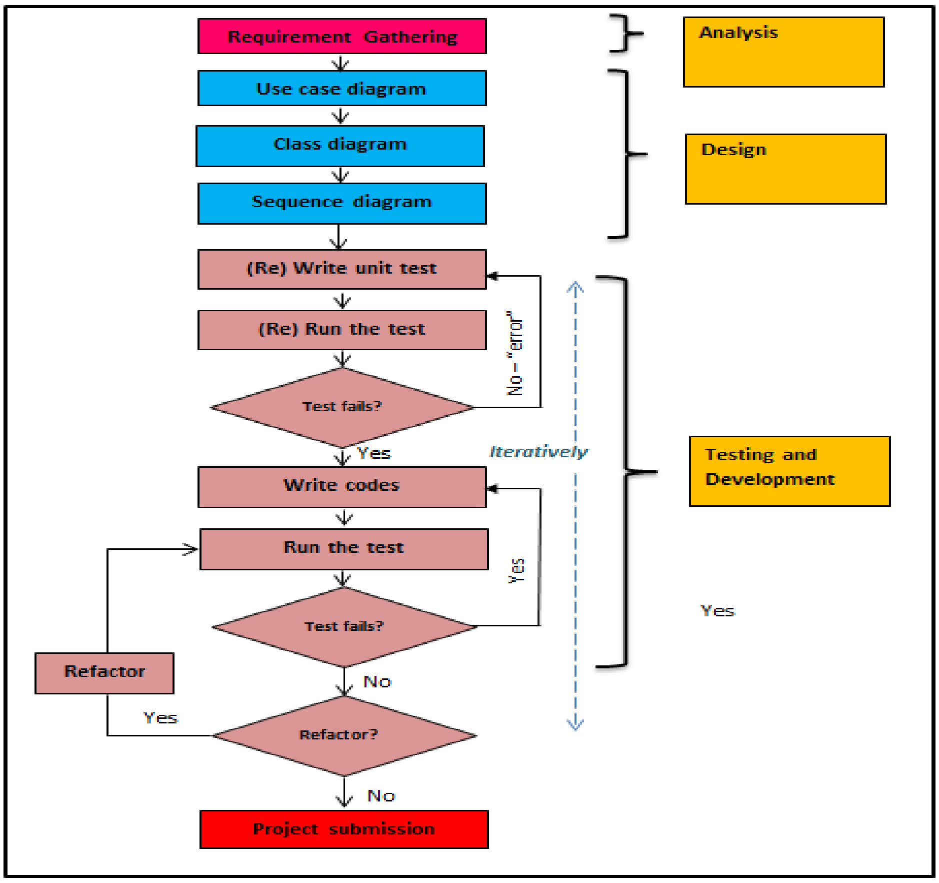


Figure 7. Proposed test first model

As a result, Test First and Test Last model introduced in this experiment have been designed based on the students' existing knowledge of development lifecycle. The model is minimizing from introducing new approach such as the user stories to the students and it allows them to elicit the JUnit test cases development, and perform the development within the boundaries in the details design. Thus, they will be able to focus on the development beside trying to understand the user stories. The definition of the terms used in Figure 7 and Figure 8 is given here:

a. Requirement gathering-the students proposed a project and describe the content of the project that they intended to develop. The proposed project must fulfil the stated criteria; auxiliary functions (array, string and mathematical computation), input/output, data manipulation and graphical user interface.
b. Use case diagram-a diagram that captures a contract between the stakeholders of a system about its behaviour. The use case describes the system's behaviour under various conditions as it responds to a request from one of the stakeholders, called the primary actor. The primary actor initiates an interaction with the system to accomplish some goal. The system responds, protecting the interests of all the stakeholders. Different sequences of behaviour, or scenarios, can unfold, depending on the particular requests made and conditions surrounding the requests. The use case collects together those different scenarios [3].
c. Class diagram-a static model that shows the classes and relationship among classes that remain constant in the system. The class diagram depicts classes, which include both behaviours and states, with relationships between the classes.
d. Sequence diagram-a model of the behaviour of objects within a use case, with focus on the time-based ordering of an activity [26].
e. Write unit test cases-the development of test cases for the respective class/method. The students have to write test cases for the auxiliary function, their systems input and output and data manipulation.
f. Write code-the development of actual production codes, which are supposed to be tested by the test cases.
g. Refactor-the process of changing a software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure [27]. It is the improvement of the executable codes (with green testing output) without changing the external behaviour. For example, the executable codes may be simplified by minimizing the duplication of existing methods.

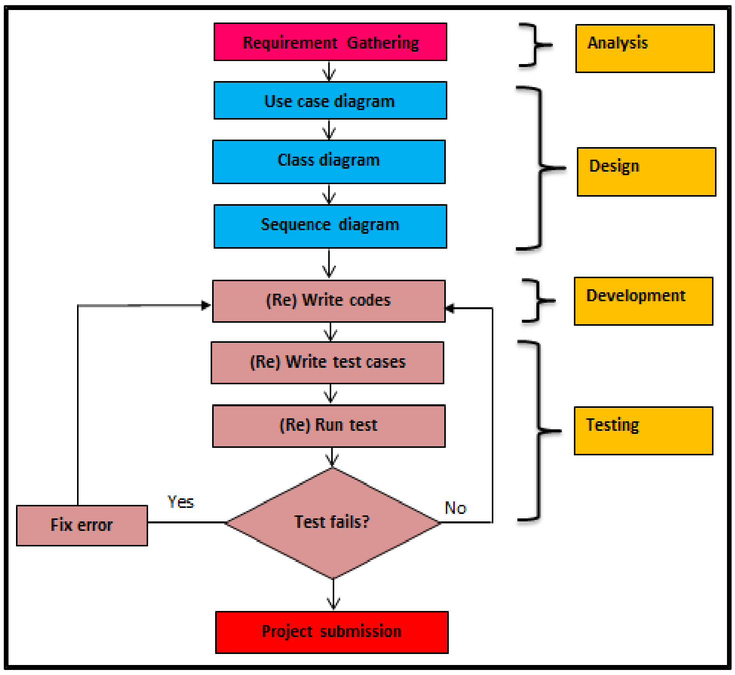


Figure 8. Proposed test last model

3.2. The Proposed Test Last Model

Test Last model depicted in Figure 7 has been designed and tailored for students' existing project implementation that is supposed to help them in executing the Test Last model. The four-steps sequence has been aligned with the Waterfall model, which consists of requirements, design, implementation, verification, and maintenance. Nevertheless, the maintenance part has been omitted from the process because the project developed by the students does not involve maintenance. The Waterfall model has also been mentioned by Balaji and Murugaiyan [28] as a model with a fixed sequence process that starts with analysis, design, and continue with the development, testing, implementation and maintenance. The development process in Test Last model is relatively similar to the Test First model. However, Test Last model does not require the students to refactor and develop their project iteratively. The students will be instructed to write the test cases after they are done with the actual codes.

4. CONCLUSION

The Agile approach, Test First, which is also known as Test Driven Development and also called as UTDD has been proven as one of the approaches that may aid students in designing complex projects and increases student confidence. Although user stories are part of Agile approach for Test First, the implementation of the existing proposed by Pančur and Ciglarič and Madeyski model has contributed to misunderstandings among students on the role of user stories beside their task to develop the test cases. In addition, the students who referred to the given user stories also informed that they were not able to identify the class or methods needed for the test cases, causing these students to even rewrite the user stories in order to fit them to their project outcomes. Because of these, the Test First model in Figure 7 has been introduced to facilitate the students in the Test First implementation by combining the students' knowledge in systems design and the existing Test First flows. More experiments and replications need to be executed based on the model to see the implementation effectiveness in producing better quality project.

REFERENCES

[1] Beck, Test-driven Development: By Example: Addison-Wesley, 2003.

[2] D. Fucci, G. Scanniello, S. Romano, M. Shepperd, B. Sigweni, F. Uyaguari, *et al.*, *"An External Replication on the Effects of Test-driven Development Using a Multi-site Blind Analysis Approach,"* in Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2016, p. 3.

[3] L. Madeyski, *Test-driven development:* An empirical evaluation of agile practice: Springer Science & Business Media, 2009.

[4] C. Desai, D. Janzen, and K. Savage, "A survey of evidence for test-driven development in academia," *ACM SIGCSE Bulletin,* vol. 40, pp. 97-101, 2008.

[5] D. Janzen and H. Saiedian, "Does test-driven development really improve software design quality?," *Ieee Software,* vol. 25, pp. 77-84, 2008a.

[6] M. Pančur and M. Ciglarič, "Impact of test-driven development on productivity, code and tests: A controlled experiment," *Information and Software Technology,* vol. 53, pp. 557-573, 2011.

[7] L. Huang and M. Holcombe, "Empirical investigation towards the effectiveness of Test First programming," *Information and Software Technology,* vol. 51, pp. 182-194, 2009.

[8] J. H. Vu, N. Frojd, C. Shenkel-Therolf, and D. S. Janzen, "Evaluating test-driven development in an industry-sponsored capstone project," in Proceedings of the Sixth International Conference on Information Technology: New Generations, 2009, p. 229.

[9] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal,* vol. 13, pp. 115-139, 1974.

[10] A. Causevic, D. Sundmark, and S. Punnekkat, *"Test case quality in test driven development: A study design and a pilot experiment,"* in Evaluation & Assessment in Software Engineering (EASE 2012), 16th International Conference on EASE, 2012a, pp. 223-227.

[11] R. Kaufmann and D. Janzen, *"Implications of test-driven development: a pilot study,"* in Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2003, pp. 298-299.

[12] D. Janzen and H. Saiedian, *"On the influence of test-driven development on software design,"* in 19th Conference on Software Engineering Education & Training (CSEET'06), 2006, pp. 141-148.

[13] C. A. Wellington, T. H. Briggs, and C. D. Girard, *"Experiences Using Automated 4ests and 4est Driven Development in Computer 9cience I,"* in Agile Conference (AGILE), *2007*, 2007, pp. 106-112.

[14] A. Causevic, D. Sundmark, and S. Punnekkat, *"Factors limiting industrial adoption of test driven development: A systematic review,"* in Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation., 2011, pp. 337-346.

[15] K. Buffardi and S. H. Edwards, *"Exploring influences on student adherence to test-driven development,"* in Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, 2012, pp. 105-110.

[16] N. Funabiki, T. Nakanishi, N. Amano, H. Kawano, Y. Fukuyama, and M. Isogai, "*Software Architecture and Characteristic Functions in Learning Management System" NOBASU"*," in Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on, 2010, pp. 109-112.

[17] V. Lappalainen, J. Itkonen, V. Isomöttönen, and S. Kollanus, *"ComTest: a tool to impart TDD and unit testing to introductory level programming,"* in Proceedings of the fifteenth annual conference on Innovation and technology in computer science education, 2010, pp. 63-67.

[18] O. A. L. Lemos, F. C. Ferrari, F. F. Silveira, and A. Garcia, *"Development of auxiliary functions: Should you be agile? an empirical assessment of pair programming and test-first programming,"* in Proceedings of the 34th International Conference on Software Engineering, 2012, pp. 529-539.
[19] S. Yenduri and L. A. Perkins, "Impact of Using Test-Driven Development: A Case Study," in *Software Engineering Research and Practice*, 2006, pp. 126-129.
[20] L. Madeyski, "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment," *Information and Software Technology,* vol. 52, pp. 169-184, 2010.
[21] W. S. Humphrey, "The Personal Software Process (PSP)," 2000.
[22] N. Juristo and A. M. Moreno, Basics of software engineering experimentation: Springer Science & Business Media, 2013.
[23] B. Rumpe, "Agile test-based modeling," arXiv preprint arXiv:14*09.6616,* 2014.
[24] R. Latorre, "A successful application of a Test-Driven Development strategy in the industrial environment," *Empirical Software Engineering,* vol. 19, pp. 753-773, 2014.
[25] R. S. Pressman, Software engineering: a practitioner's approach: Palgrave Macmillan, 2005.
[26] A. Dennis, B. H. Wixom, and D. Tegarden, Systems Analysis and Design with UML: Wiley, 2009.
[27] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code: Pearson Education, 2012.
[28] S. Balaji and M. S. Murugaiyan, "Waterfall vs. V-Model vs. Agile: A comparative study on SDLC," *International Journal of Information Technology and Business Management,* vol. 2, pp. 26-30, 2012.

BIOGRAPHIES OF AUTHORS

Normi Sham Awang Abu Bakar is an assistant professor in the Department of Computer Science, International Islamic University Malaysia. She obtained her PhD in Computer Science at the Australian National University. Her research interests are in the area of empirical software engineering, open source quality, agile methodology, mining software repositories and software engineering education.

Norzariyah Yahya is lecturer of Centre for Foundation Studies, IIUM since 2003 and postgraduate from Kuliyyah of Information and Communication Technology, IIUM. Her research interest is in Software Engineering and Agile specifically.