International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume: 3 | Issue: 3 | Mar-Apr 2019 Available Online: www.ijtsrd.com e-ISSN: 2456 - 6470

Unit Test using Test-Driven Development Approach to Support Reusability

Myint Myint Moe

University of Computer Studies (Hpa-An), Kayin State, Myanmar

How to cite this paper: Myint Myint Moe "Unit Test using Test-Driven Development Approach to Support Reusability" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-3 | Issue-3, April 2019, pp.194-196, URL: http://www.ijtsrd.com/papers/ijtsrd21731.pdf

IJTSRD21731

ABSTRACT

Unit testing is one of the approaches that can be used for practical purposes in improving the quality and reliability of software. Test-Driven Development (TDD) adopts an evolutionary approach which requires unit test cases to be written before implementation of the code. TDD method is a radically different way of creating software. Writing test first can assure the correctness of the code and thus helping developer gain better understanding of the software requirements which leads to fewer defects and less debugging time. In TDD, the tests are written before the code is implemented as test first. The number of defects reduced when automated unit tests are written iteratively similar to test driven development. If necessary, TDD does the code refactoring. Refactoring does to improve the internal structure by editing the existing working code, without changing its external behavior. TDD is intended to make the code clearer, simple and bug-free. This paper focuses on methodology and framework for automation of unit testing.

Keywords: *Unit Testing, TDD, Refactoring*

I. INTRODUCTION

Unit Testing finds the defects in each and every unit of the application at initial level of testing. Unit testing ensures that code works correctly and improves the quality, reliability and cost of the application software development [9]. TDD starts with designing and developing tests for every small function of an application. In TDD approach, first, the test is developed which specifies and validates what the code will do. In the normal testing process, first generate the code and then test. Tests might fail since tests are developed even before the development. In order to pass the test, the development team has to develop and refactors the code. Refactoring is a powerful agile technique for improving existing software [7]. Refactoring a code means changing some code without affecting its behavior. The idea of refactoring is to carry out the modifications as a series of small steps without introducing new defects into to the system. By re-running the test cases, the developer can be confident that code refactoring is not damaging any existing functionality [6].

This paper is organized with (5) sections. The brief introduction is described in section I. Objectives, Related work, Methodology are presented in II, III and IV respectively. The paper is concluded in section V.

II. Objectives

Software testing is an essential part of software development process to assure the quality of software systems [5]. Unit testing is to validate that each unit of the software performs as designed. Unit testing is a practice that forces to test small, individual and isolated units of code. [7]. TDD is to write and correct the failed tests before writing new code (before development). TDD helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. TDD is a process of developing and running automated test before actual development of the application. Refactoring is a well-defined process that improves the quality of systems and allows developers to repair code that is becoming hard to maintain, without throwing away the existing source code and starting again [6].

III. Related Work

In [2], the authors proposed on the Effectiveness of Unit Tests in Test-driven Development. Author conducted an experiment in an industrial setting with 24 professionals. Professionals followed the two development approaches to implement the tasks. Author measures unit test effectiveness in terms of mutation score. Author also measures branch and method coverage of test suites to compare author's results with the literature. As a result, In terms of mutation score, author has found that the test cases written for a test-driven development task have higher defect detection ability than test cases written for an incremental test-last development task. Subjects wrote test cases that cover more branches on a test-driven development task compared to the other task.

However, test cases written for an incremental test-last development task cover more methods than those written for the second task.

In [3], the authors described using Test-Development Development to Improve Software Development Practices. Author focused on the participants' point of view and experience of using TDD in software development. The data analysis of author will be described along with a conclusion on how TDD can improve software development. As a result, TTD has the ability to help practitioners throughout the software development implement phase in many ways. TDD was analyzed with five factors in mind and their outcomes compared to this research which showed very similar results. The main advantages of using TDD can be outlined as: improved code quality, less defects, easier maintenance, safety net and reliable software.

In [5], authors proposed An Industrial Evaluation of Unit Test Generation: Finding Real Faults in a Financial Application. As the method, RANDOOP is one of the most stable random test generation tools, with easy to follow instructions to get it up and running in short time. EVOSUITE is a search-based unit test generation tool for Java that uses a genetic algorithm to evolve a set of test cases with the intention of maximizing code coverage. As a result, on average, RANDOOP can find faults in 36.8% of the runs, whereas EVOSUITE in 50.8% of the runs. Test generation tools detected up to 56.40% (EVOSUITE) and 38.00% (RANDOOP) of faults in all executions.

IV. Methodologies

Test-Driven Development (TDD) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, and then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements [6]. In Test Driven Development, the developer writes automated unit tests for the new functionality they are about to implement. It is a software engineering process that follows small development cycle. The automated unit tests for any application can be written in two ways: Before code implementation and after code implementation. If unit tests are written before any code implementation then it is known as Test Driven Development. If Unit tests are written after the code implementation then it is known as Test after Development. [9]

A. Proposed System

Test driven development model cycle consists of:

Writing a Test: A unit test (manual or automated, preferably automated) is first written to exercise the functionality that is targeted for development. Before writing the test, the developer is responsible for understanding the requirements well. A unit test shall also contain assertions to confirm the pass/fail criteria of the unit test.

Run to fail /make it compile: Since the feature is yet to be implemented, the unit test that was written in Step 1 is bound to fail. This step is essentially a validation step for the unit test written, as the test shouldn't pass even if there is no code written for it. Often unit tests are automated and there are chances that the tests fail because of syntax or compilation errors. Sanitization of the tests by removing these errors is also an essential part of this step.

Implementing (complete /partial) functionality:
This step involves developing the part of the functionality for which the unit test is written and will be validated.

Making tests to pass: Once the unit tests for the developed code have passed, the developer derives confidence that the code fulfills the requirements.

Code refactoring: The unit tests might have passed, but code refactoring may still be required for reasons including handling errors elegantly, reporting the results in the required format, or carving a subroutine out of the written code for re-usability.

Repeating the cycle: The unit test/set of unit tests is/are refactored to cater to new functionality or push towards completion of the functionality. [11]

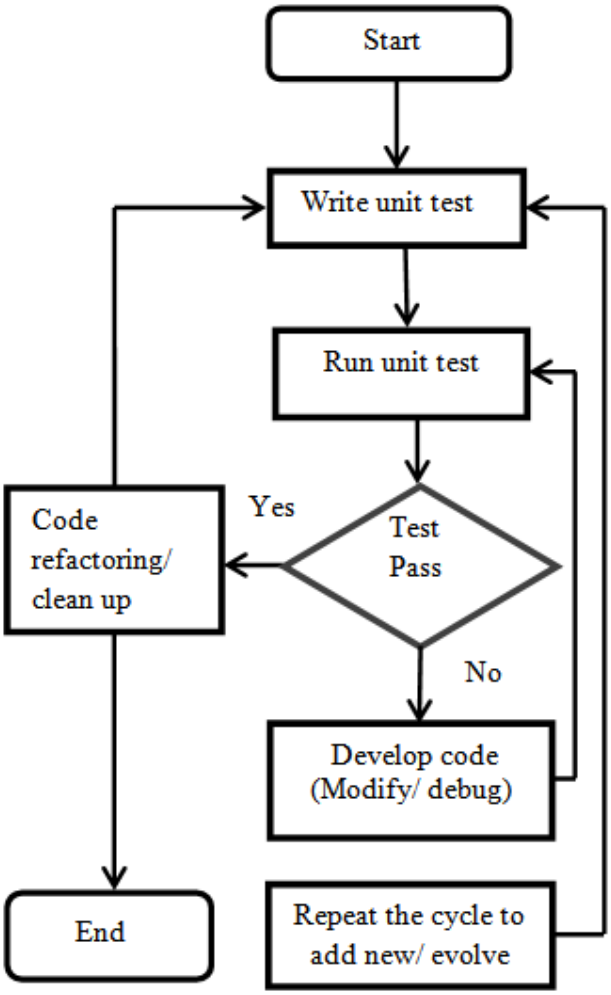

Figure1: TDD cycle

The proposed system verifies the correctness of all codes and gives developers confidence by reducing code complexity providing that it is used persistently over time and motivates developers to produce higher code quality. By using proposed system, developers were more productive and make less effort per line- of code. In traditional development, developers develop their code first, complete the functionality and go for manual testing. Traditional development, first code is written and then code is tested. If tests are written after the implementation, there is a risk that tests are written to satisfy the implementation. Development time is relatively high in proposed system. It nearly takes more time than Test-Last Development or

traditional development. This is because of its iterative process between testing, coding, refactoring. As a future plan, we will reduce development time and it will be less time- consuming, using proposed system. Besides, the author also thinks if proposed system could be appealing to students, teaching approach should be applied in higher education.

B. Advantages and Disadvantages of TDD

Advantages of TDD, shortens the programming feedback loop. TDD promotes the development of high-quality code. User requirements more easily understood. TDD reduced interface misunderstandings and software defect rates. . TDD provides concrete evidence that your software works. TDD gives batter Code and less Debugging Time.

Disadvantages of TDD, Programmers like to code, not to test. Test writing is time consuming. Test completeness is difficult to judge. TDD may not always work [10]

C. Comparison of TDD and Traditional Testing

TDD approach is primarily a specification technique. It ensures that your source code is thoroughly tested at confirmatory level. With TTD, a successful test finds one or more defects. When a test fails, you have made progress because you know that you need to resolve the problem. With traditional testing, a successful test finds one or more defects. When a test fails, you have made progress because you know that you need to resolve the problem. TDD ensures that your system actually meets requirements defined for it. It helps to build your confidence about your system. In TDD more focus is on production code that verifies whether testing will work properly. In traditional testing, more focus is on test case design. In order to fulfill requirements, the test will show the proper/ improper execution of the application. In TDD, achieve 100% coverage test. Every single line of code is tested, unlike traditional testing. The combination of both tradition testing and TDD leads to the importance of testing the system rather than perfection of the system. [6] [10].

V. Conclusion

In this paper**,** the proposed system is a development approach that can help developers to design a code and supports developers in their work with confidence. Consequently, they will be able to write more reliable software. This system has given the developers deeper logical understanding of their code and has supported them to improve their development skills. It counts the bugs and defects over a period of time. This approach enables thorough unit testing which improves the quality of the software and enhances customer satisfaction. The proposed system is an effective tool to improve productivity in long run and can significantly reduce the defect density of developed software either immediately or in the long run. Unit tests detect changes that may break a design contract. They help with maintaining and changing the code. Unit testing reduces defects in the newly developed features or reduces bugs when changing the existing functionality. That is why, Unit testing is chosen than other testing in this research. This system has the ability to speed up the process.

References

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

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

[3] RAQUELITA ROS AGUILAR, "Using Test-Development Development to Improve Software Development Practices", REYKJAVIK UNIVERSITY, SPRING 2016, T-622-UROP.

[4] Arcuri, A., Campos, J., & Fraser, G. (2016) "Unit Test Generation during Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins" 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST).

[5] Almasi, M. M., Hemmati, H., Fraser, G., Arcuri, A., & Benefelds, J. (+2017). "An Industrial Evaluation of Unit Test Generation: Finding Real Faults in a Financial Application" 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP).

[6] 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).

[7] Authors : 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.info.

[8] www.JUnit.org

[9] 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

[10] Author: Mark Levison; published : Oct 14, 2008, www.agilepainrelief.com, www.guru99.com