Volume 3, Issue 4, April 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Review on Test Driven Development: a state of practice

Kavita*	Swapnika	Vinod Saroha	Sarika	Jyoti
SES, BPSMV	*SES,BPSMV*	*Lecturer,BPSMV*	*SES,BPSMV*	*SES,BPSMV*
Khanpur kalan, Sonepat	*Khanpur kalan , Sonepat*	*Khanpur Kalan, Sonepat*	*Khanpur,Sonepat*	*Khanpur,Sonepat*
Haryana, India.	*Haryana, India.*	*Haryana, India.*	*Haryana, India.*	*Haryana, India.*

Abstract— Test driven development (TDD), also known as Test-First coding which is a practice where programmers write a production code only after writing a automated failing test case. This practice is primarily used in software development circle. How much acceptance has it gained in its short life span? This is explained in this paper.

Keywords— Test Driven Development, agile software development

I. INTRODUCTION

Test driven development was firstly used in NASA's project mercury in the year 1960's which employed test first practices in the development of spaceship's software. This is a practice where programmer first writes test cases and then writes production code. This approach offers a completely opposite view of the traditional test-last approach. In test last approach, first production code is applied and then one test code is applied to exercise it. Test first coding is easily done with modern IDE's where one can write test and production code and get immediate feedback. Kent Beck described TDD (test driven development) in his book: *Extreme Programming Explained: embrace change in 1999.* He describes being a child and reading programming book that describes test-first approach, where programmer first produces the expected output and then write code until actual output did not match with expected output. Test driven development does not provide guidance for its implementation. An initial instruction from Beck was as " never write a line of production code without a broken test case".

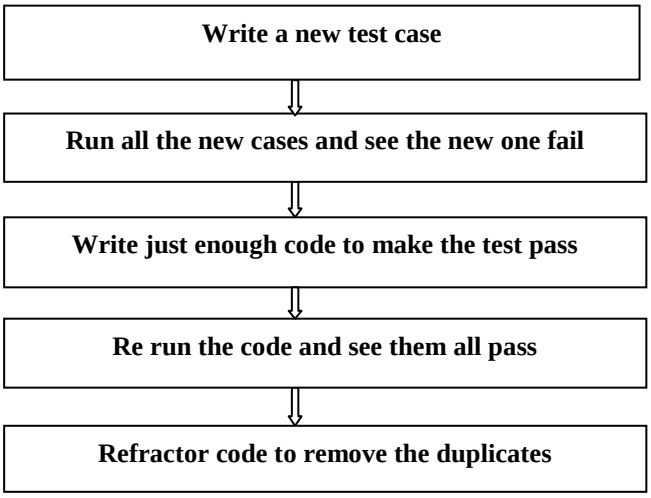
(A). ANALOGY TO TRAFFIC LIGHT

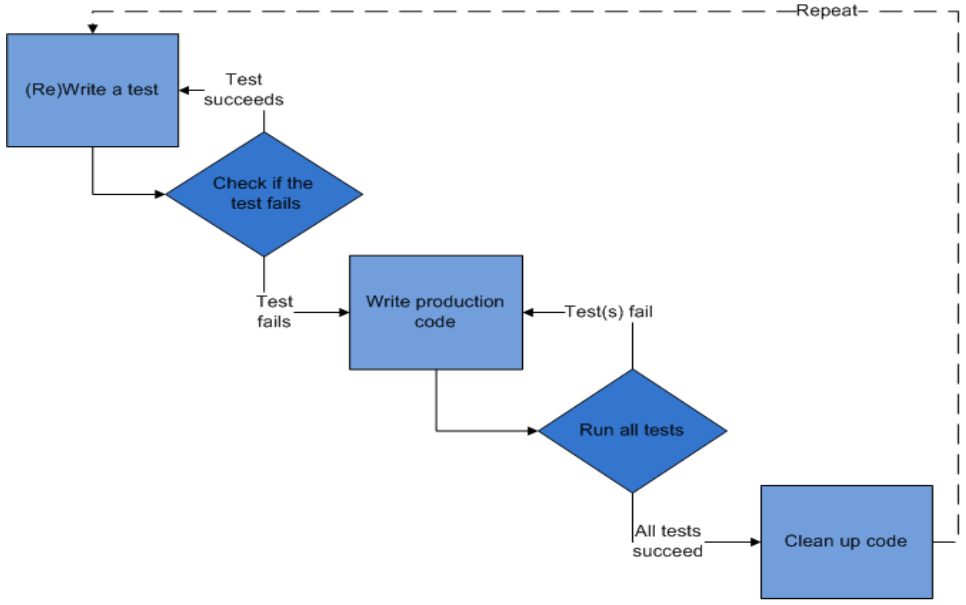
William Wake gave an idea of how TDD approach should work by concept of traffic-light metaphore. Example followed by him uses *green, yellow and red* pattern:

- Start to write the test gives initial *green* light.
- Test failed to compile because no production code is available, results in *yellow* light.
- Once a stub was written for the production code and test fails as stub did not do anything. For example- if we call any function in C or C++ language but function is not declared anywhere in the code then compilation error will be shown i.e. test fails, results in *red* light.
- Once the production code written and test case passed, then again a *green* light status will return to programmer.

(B) RED, GREEN AND REFRACTOR: A FIVE STEP PROCESS

- The red light represents the failing or non compiling test code.
- Green light represents the minimum amount of code, required to pass the test code.
- Refractor is used to remove code duplication.
- Refractor is used to remove code duplication.





II. TDD EXTENSIONS

II.I AGILE SPECIFICATION DRIVEN DEVELOPMENT:

This is a technique that combines both- TDD and Design by Contract (DbC). DbC –this allows a language to declare invariants for the class and pre and post condition for a method. It was given by Bertrand and Meyer. Microsoft has introduced their version of DbC into .NET framework, called code contracts. Meyer has also introduced an approach called *Quality-first Model* to software development. Another developer described Quality approach as:-

i. Write code as soon as possible because then supporting tools immediately do syntax, type and consistency checking.
ii. Current set of functionality should be working before moving to next. Abnormal cases must be dealt with. Interwine analysis, design and implementation.
iii. Always have a working system. Get cosmetic and style right.

The DbC approach is at core of Quality-first model.

Similarities in DbC and TDD:
In both the approaches one unit of functionality must be finished before moving to next.

Differences: in TDD, it asks the developer to focus on the most common case, whereas DbC expects the developer to focus on abnormal cases first.

II.II Behaviour driven design (BDD):

BDD was first introduced by Dan North. Test-driven development name was replaced by Behaviour driven development. The reason behind it is to clear the confusion that whether TDD is a testing method or design method. By changing the developer's focus to the behaviour of code. It sets its mind away from design technique.

Life cycle for learning and adoption of TDD:

- Starts writing unit test around code. Increased sense of confidence with increasing body of tests.
- Focus on writing only the necessary code. Notice that tests serve as documentation to the working of code and realize that test assist in "discovering" the API.
- Realizes that TDD is about defining behaviour.

New testing framework has developed that change the nomenclature, to get developers to think beyond the viewpoint that TDD is about testing. Change in nomenclature based on- Sapir Whorf theory that gives statement "language used influences your thought". *For e.g.* instead of using testing terminology "assertion" the framework uses "ensure that" in JBehave (java based framework) that focuses on the behaviour of code. For ruby, 'assertion' is replaced with sentence type structure where actual object is the subject and assertion statement is the verb: like-'actual.should.equal expected'.

Key aspects of BDD include:

i. The word 'test' should not be included in the test name.
ii. Test method name should be sentences and must begin with 'should', for e.g. 'should take a cup of tea whenever get cold'.

iii. Test should focus on behaviour. This makes it easier to change the structure as classes grow and when broken into separate classes.

II.III **Acceptance Test Driven Development:**

Traditional TDD meets the low-level requirements of the project. It focuses on the smallest tests that could possibly be written. Developers first write the failing test case and then write production code until test case passes. But there is no mechanism for developers to put their requirements in context. So we can say that in traditional TDD, code quality was better but did not meet the high level requirements of the project.

For high level requirements to meet, software engineers are focusing to set the context for overall development. Beck gives the concept of application test driven development, where users would write the test by themselves. It allows the users to write acceptance test that is used by the developers to see if correct functionality is provided by their software. BDD and ATDD are two approaches.

A FIT table approach was introduced in 2002. Acc to this, the customers enters test data into a table using word processor and then translate the data into a standard FIT fixture through FIT client and server process. This will increase the running time of program. Beck objects to ATDD because tests would not be under the control of developers. He objects to the lag in time between test and feedback. He predicts that users and organization will not be able for the software development in a timely manner.
ATDD would create delay and lose needed control.

II.IV **Growing Objects, using tests:**

In this, developers begin with a functional test case that is derived from a system requirement. In this approach, *tests are written by developers not by end users so this differs from ATDD.* One more requirement for this is the clear understanding of the user stories to create the correct test.

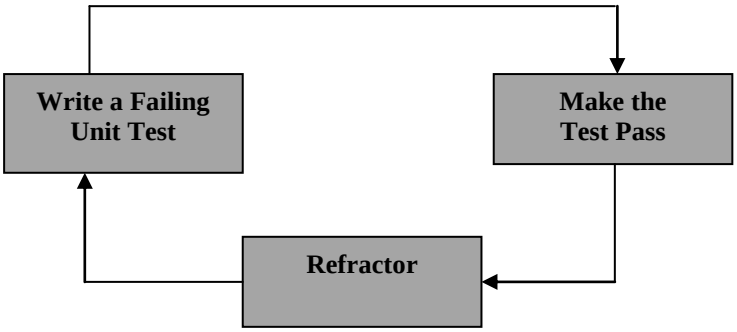


Fig. Fundamental TDD Cycle

By adding the functional test the cycle now looks like-

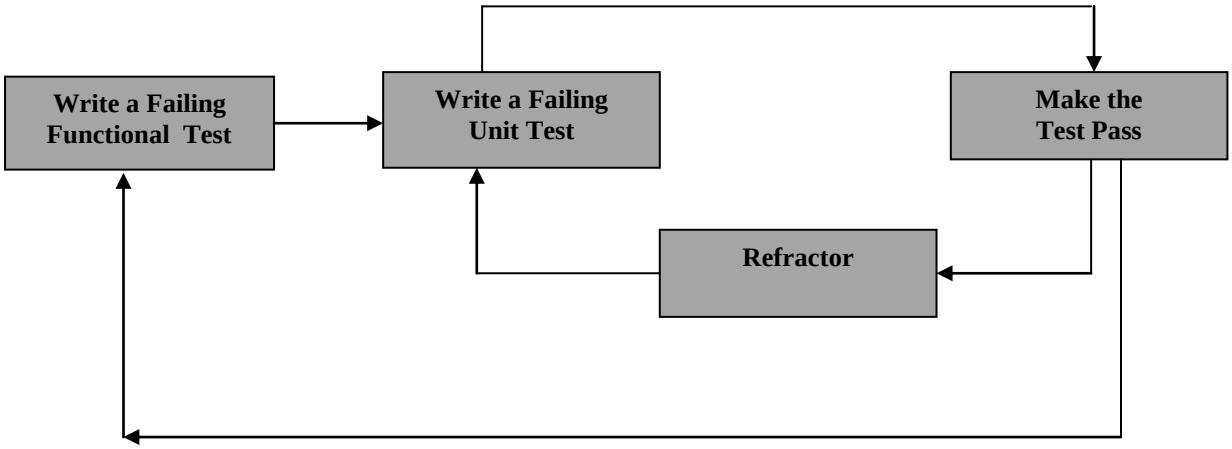


Fig. Functional TDD Cycle

III. FUTURE WORK

Particularly TDD relates to two significant questions:-

- How does the developer know that he is developing correct functionality?
- When does the developer know that he is finished?

A new approach is proposed for these issues: ***Means-Ends Analysis Approach (MEAA):***

It originates in AI area. In this approach, a current state and a goal state are defined. An action is choosed to reduce the differences between two. MEAA uses functional test case to represent the goal state. When this test case passes then development of functional system requirement is completed. A traditional approach is needed to develop new components. Unit level test of new components become intermediate goal. The remaining user stories remains on "To-Do list" (until first functional test passes completely).

Future work on the MEAA includes a complete codification of Mean-End Analysis Approach. This will search for "more precise definition" of TDD. Experiments will be performed to determine if MEAA leads the developer to a better coverage of requirements.

REFERENCES

[1] Beck, K. 2000. *Extreme Programming Explained.* Addison-wesley.

[2] Beck, K. 2001. Aim, Fire. *Software*, 18(5):87-89, Sept.-Oct. 2001.

[3] Larman, C., & Basili, V. R. 2003. Iterative and Incremental Development: A Brief History. *Computer* (June 2003), 47-56

[4] Beck, K. 2003. *Test-Driven Development: By Example* Addison-Wesley Professional.

[5] Astels, D. 2003. *Test-Driven Development: A Practical Guide* . Pearson Education, Inc., Upper Saddle River, NJ.

[6] Wake, W. 2001. The Test-First Stoplight http://xp123.com/articles/the-test-first-stoplight/

[7] Ambler, S. 2010. How Agile Are You? 2010 Survey Results http://www.ambysoft.com/surveys/how Agile Are You2010.html

[8] West, D., & Grant, T. 2010. Agile Development: Mainstream Adoption Has Changed Agility. Cambridge: Forrester.

[9] Desai, C., & Janzen, D. S. 2008. A Survey of Evidence for Test-Driven Development in Academia. *inroads – SIGCSE Bulletin , 40* (2), 97-101.

[10] Desai, C., & Janzen, D. S. 2009. Implications of Integrating Test-Driven Development into CS1/CS2 Curricula *SIGCSE'09* (pp. 148-152). Chattanooga, TN: ACM.

[11] Kollanus, S., & Isomottonen, V. 2008. Test-Driven Development in Education: Experiences with Critical Viewpoints. *ITiCSE* (pp. 124-127). Madrid, Spain: ACM.

[12] Koskela, L. 2008. Test Driven: Practical TDD and Acceptance TDD for Java Developers. Greenwich, CT: Manning Publications Co.

