Test driven development: Growing Object using test

Nitin[1], Shubha Jain[2]
[1,2]Department of Computer Science & Engineering, Kanpur Institute of Technology, Kanpur (India)

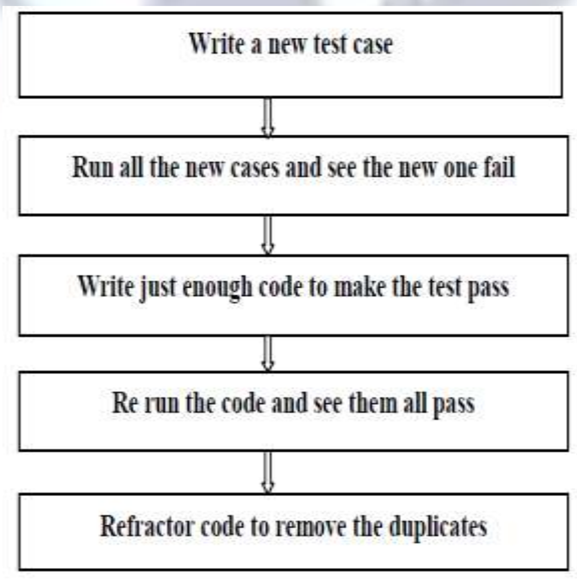
Abstract: Test driven development (TDD), also known as Test-First coding which is a practice where programmers write a production code only after writing a automated failing test case. This practice is primarily used in software development circle. In Growing Objects, using tests we implement Component Testing.

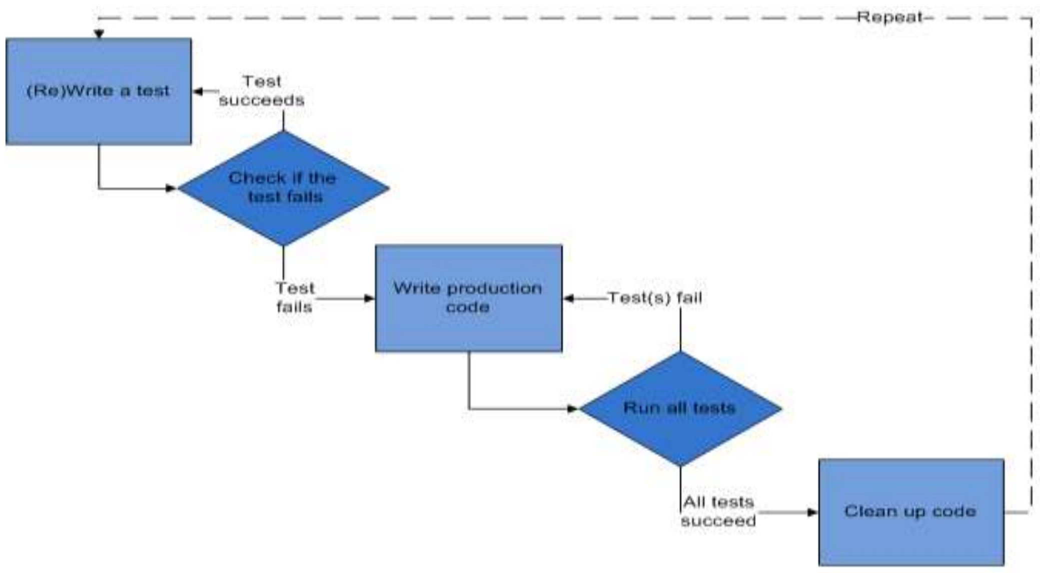
Keywords: Test driven development, Test first coding, agile specification driven development, behaviour driven design, acceptance test driven development, Growing object using test.

Introduction

Test-first programming has been practiced at least from the 1960s. However, these ideas have become well known in the software engineering community just during the last decade in the form of Test –Driven development (TDD).TDD was first popularized as one of the key practices in Extreme Programming (XP), but later it has also been discussed as a stand alone process. The basic idea of TDD is simply to write tests before code in small iterations. First, developer writes a test case that is just enough to define the next functionality. The next step is to write code that is first enough to pass the test. Finally, The code is refactored, if needed. These step are iterated in short cycles through the whole development process. Originally TDD was introduced as a development, not a testing, method. In several as a development, not a testing, method. In several studies, TDD has been suggested to provide a variety of different benefits, such as better productivity, better quality and high test coverage. Some studies also suggest that TDD may improve program design and developers' confidence on their code.

However, the current empirical evidence on TDD includes many contradictory results. Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.


Write a new test case
Run all the new cases and see the new one fail
Write just enough code to make the test pass
Re run the code and see them all pass
Refactor code to remove the duplicates



TDD EXTENSION

A. Agile Specification Driven Development

We present an agile approach to Specification-Driven Development, which combines features of Test-Driven Development and the plan based approach of Design-by-Contract. We argue that both tests and contracts are different type of specifications, and both are useful and complementary for building high quality software. We conclude that it is useful for being able to switch between writing tests and writing contracts, and explain how Specification-Driven Development support this capability. Agile approach to Specification-Driven Development, Which combines features of Test-Driven Development and the plan-based approach of Design-by-Contract. Both tests and contracts are different quality software.

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.

B. Behaviour Driven Design (BDD)

Behavior-driven development (abbreviated BDD) is a software development process based on test-driven development (TDD). Behavior-driven development combines the general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design to provide software developers and business analysts with shared tools and a shared process to collaborate on software development, with the aim of delivering "software that matters". Although BDD is principally an idea about how software development should be managed by both business interests and technical insight, the practice of BDD does assume the use of specialized software tools to support the development process. Although these tools are often developed specifically for use in BDD projects, they can be seen as specialized forms of the tooling that supports test-driven development. The tools serve to add automation to the ubiquitous language that is a central theme of 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 behavior 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".

C. 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.

D. GROWING OBJECT USING TEST

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.

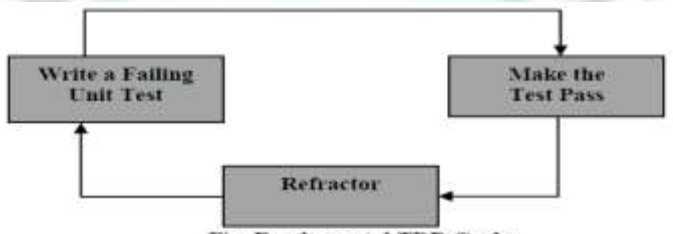


Fig. Fundamental TDD Cycle

By adding the functional test the cycle now looks like

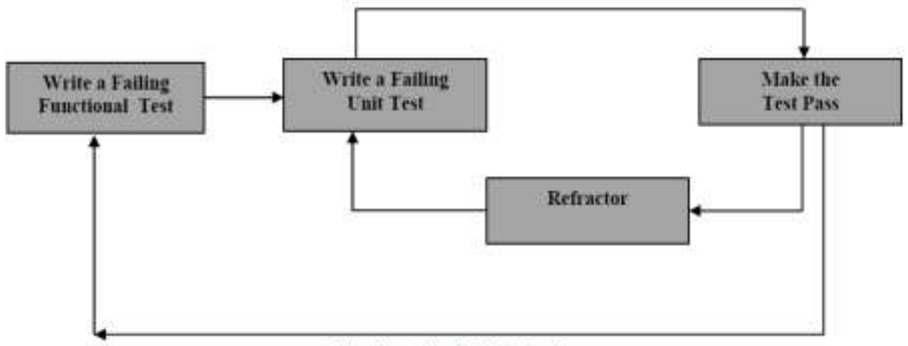


Fig. Functional TDD Cycle

We are implementing the Component test than cycle now look like

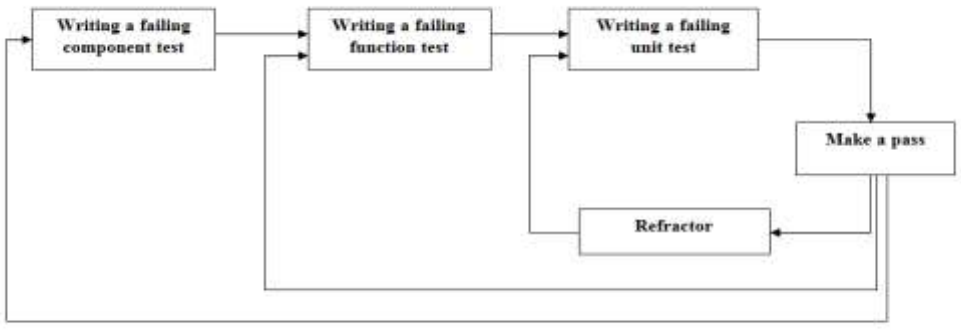

Writing a failing component test
Writing a failing function test
Writing a failing unit test
Make a pass
Refactor

Figure 3: Component TDD cycle

FUTURE SCOPE

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.

CONCLUSIONS

In this paper we work Test driven development: Growing object using test. In Growing object using test, programmers does not use Component test. We implement Component test in Growing object using test. If Component test is fail we use Figure 3. Component TDD cycle for correction on a software.

REFERENCES

[1]. Kavita et al., International Journal of Advanced Research in Computer Science and Software Engineering 3(4), April - 2013, pp. 429-432.
[2]. Beck, K. 2003. Test-Driven Development: By Example Addison-Wesley Professional J. Breckling, Ed., The Analysis of Directional Time Series: Applications to Wind Speed and Direction, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.
[3]. Koskela, L. 2008. Test Driven: Practical TDD and Acceptance TDD for Java Developers. Greenwich, CT: Manning Publications Co.
[4]. Desai, C., & Janzen, D. S. 2008. A Survey of Evidence for Test-Driven Development in Academia. inroads – SIGCSE Bulletin , 40 (2), 97-101S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, "A novel ultrathin elevated channel low-temperature poly-Si TFT," IEEE Electron Device Lett., vol. 20, pp. 569–571, Nov. 1999.
[5]. Desai, C., & Janzen, D. S. 2009. Implications of Integrating Test-Driven Development into CS1/CS2 Curricula SIGCSE'09 (pp. 148-152). Chattanooga, TN: ACM M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, "High resolution fiber distributed measurements with coherent OFDR," in Proc. ECOC'00, 2000, paper 11.3.4, p. 109.
[6]. http://en.wikipedia.org/wiki/Test-driven_development.
[7]. http://link.springer.com/chapter/10.1007%2F978-3-540-24853-8_12#page-1.
[8]. http://en.wikipedia.org/wiki/Behavior-driven_development.