ICIC Express Letters
Volume **11**, Number **9**, September **2017**
ICIC International ©2017 ISSN 1881-803X
pp. 1435–1440

INTEGRATING TEST DRIVEN DEVELOPMENT INTO THE SOFTWARE DEVELOPMENT PROCESS

Haiyan Zhu, Qingcong Lv and Xiaohua Liu

School of Computer Science and Technology
Shandong Institute of Business and Technology
No. 191, Binhai Middle Road, Yantai 264005, P. R. China
hzhu5@hotmail.com

Received April 2017; accepted June 2017

Abstract. *Test Driven Development (TDD) is a software development method which requires developers to write low-level tests before coding. It has become popular in recent years. Several studies have been conducted to analyze the influence of TDD on software quality and productivity. In this paper, we investigate the effectiveness of TDD from the process view. If TDD is considered as a software development subprocess, it can be applied with other process models. As an example, we show how the waterfall model and prototyping can be melded with TDD to create a new process model. Our experimental results suggest that some improvements in quality can be achieved by incorporating TDD into a software development process.*
Keywords: Test driven development, Software development process, Software quality, Software construction

1. **Introduction.** Extreme Programming (XP) is a new software development method proposed by Beck [1]. Test Driven Development (TDD) is an essential part of the XP development process. Many developers believe that TDD brings improvement in code quality and productivity. Although not all programmers agree with all of the XP practices, TDD has become popular and is being widely adopted in industry.

The main idea of TDD is that it requires developers to write low-level functional tests before coding [2-4]. It is also referred to as Test-First Development. Test Driven Development reverses the order of implementation and testing compared to a traditional software development. It involves the repetition of a very short development cycle: writing a test, producing code to pass the test and refactoring the code.

Several studies have been conducted to analyze the influence of TDD on software quality and productivity [5-10]. The results of a case study of an IBM team who has been practicing TDD for five years were reported in [5]. A controlled experiment for evaluating the effectiveness of TDD was conducted in [6]. Crispin reported experiences with her XP team and explained how TDD improved code quality [7]. Another case study can be found in [8]. Rafique and Misic provided a systematic meta-analysis of 27 studies for the impact of TDD on external code quality and productivity [9]. Guerra showed how TDD was done in an agile environment [10]. All these studies showed that TDD had a positive effect on the software quality, but they focused on TDD as a single process model. In this paper, we investigate the effectiveness of TDD from the process view. We capture the development process as a collection of process models, rather than focusing on a single model. We consider TDD as a subprocess and integrate it into a software development process. Our results show that incorporating TDD into a software development process, some improvements in quality can be achieved. Section 2 reviews the concept of the test driven development. Section 3 discusses the software development process and the test driven development as a subprocess. Section 4 explains the advantages of TDD. Section 5 presents a simple experiment. Section 6 shows our conclusions.

2. **TDD Development Cycle.**

TDD consists of the following steps.

1) Add a test. Each test adds some degree of functionality. The developer must understand feature's requirements before writing the test.

2) Run all tests and see if the new one fails. The new test does not pass because the feature has not been implemented.

3) Write code to satisfy the new test. The new code will cause the test to pass.

4) Run all automated tests, repeat 3 if necessary until all tests pass.

5) Occasionally refactor to clean up dirty code and improve code structure. Run all tests after refactoring to ensure all tests pass. The developer needs to make sure that refactoring is not damaging any existing functionality.

Figure 1 illustrates the TDD development cycle.

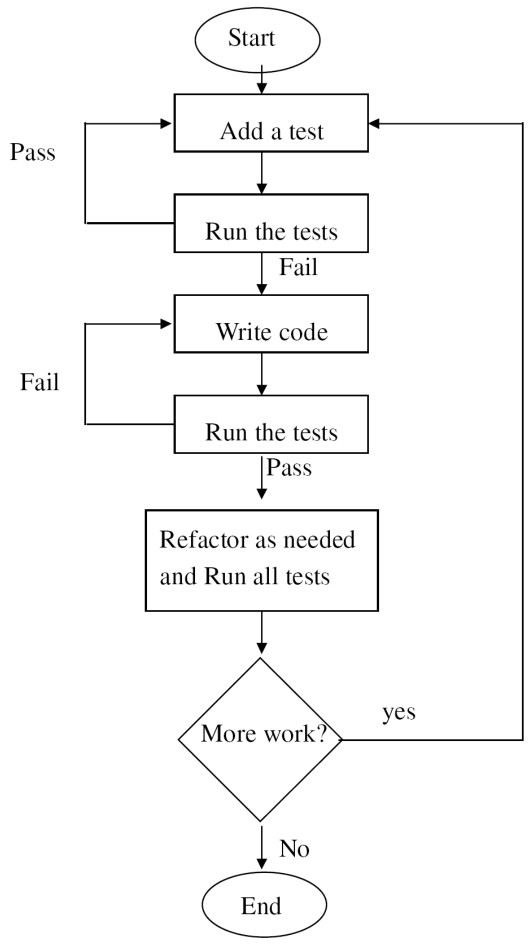

FIGURE 1. The TDD development cycle

TDD is not a testing technique, but a test-driven development method. It involves writing and executing test cases which are a description of the software design. By developing test cases and code, the design evolves and the code is gradually refactored to make the design cleaner and code structure better. Thus many people call TDD a design technique. In fact, it melds program design, implementation and testing in a series of very short iterations. It focuses on simplicity and feedback and achieves incremental development.

The basic rule of TDD is that test cases are written for a new feature before implementing code. When writing a test, the developer must understand the correct behavior for the component or the system, but he does not have to care about implementation details. For a new feature, there are often many test cases that have to be written down. The testing framework such as xUnit (JUnit for Java) plays an important role in TDD. It helps developers to create, execute and manage all these test cases. Programmers can create tests easily and execute a suite of test cases with a single button click.

3. Software Development Process and TDD.

3.1. Software development process.

A software development process describes the life of a software product from its conception to its implementation, delivery, use and maintenance [11]. It defines the order, control and evaluation of the tasks involving activities, constraints and resources. Processes are important because they impose consistency and structure on a set of activities. Using a process developers know how to do something well and when to do it.

In theory, there are different process models such as waterfall model, transformational model, spiral model and agile methods [11]. Each model has its own characteristics, benefits and drawbacks. For a new project, managers need to understand the goals, constraints and unique feature of the project. Then they carefully choose the process model to reflect the development goal. Usually the chosen process model should be tailored for the special situation. At last they create their own process model for the project.

3.2. TDD subprocess.

Although TDD is the core practice of the Extreme Programming (XP) development process, it can be used with other process models. We can view TDD as a subprocess, which is applied within the context of different projects to form a new process model.

The waterfall model illustrated in Figure 2 is a traditional process model. It describes software development activities in a linear sequence: requirement analysis, system design,

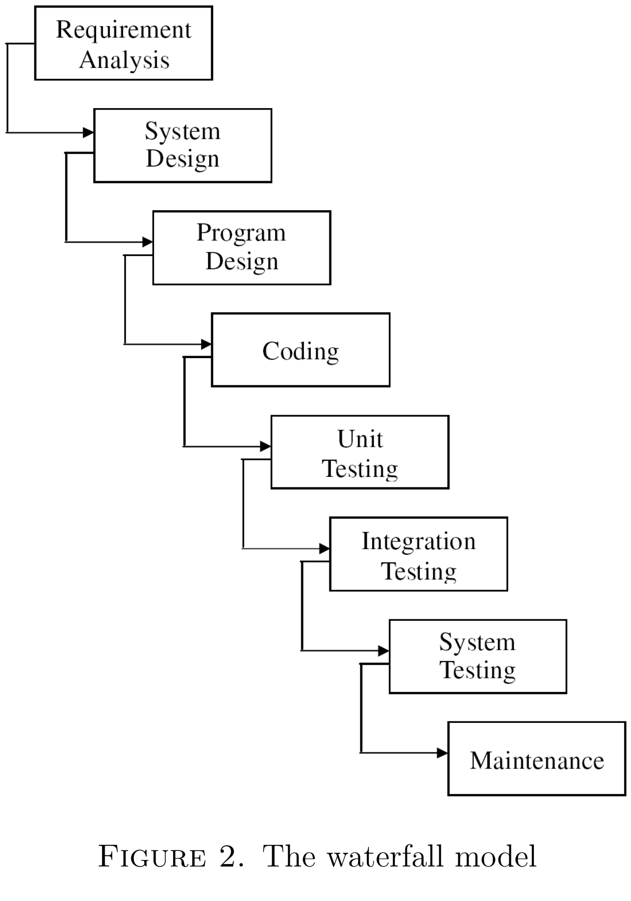

Figure 2. The waterfall model

program design, coding, unit testing, integration testing, system testing and maintenance. There are some drawbacks with the waterfall model. The waterfall model does not tell developers how to handle changes during software development. It does not reflect the way code is really developed. Software is usually developed with a great deal of iteration.

TDD involves small cycles of iterations, writing test, implementing that test and refactoring the code. TDD can be used in program design and coding stages to overcome some disadvantages of the waterfall mode.

To improve the quality of requirement analysis and design, the prototyping subprocess can be used in the waterfall model. For example, developers may build a small system to implement some key requirements to ensure that the requirements are correct, feasible and practical; if not, revisions are made until there is common agreement on what the system should do. Similarly, system design may also be prototyped to help developers assess different design strategies and decide which is best for a particular project.

The modified waterfall model with prototyping and TDD is shown in Figure 3.

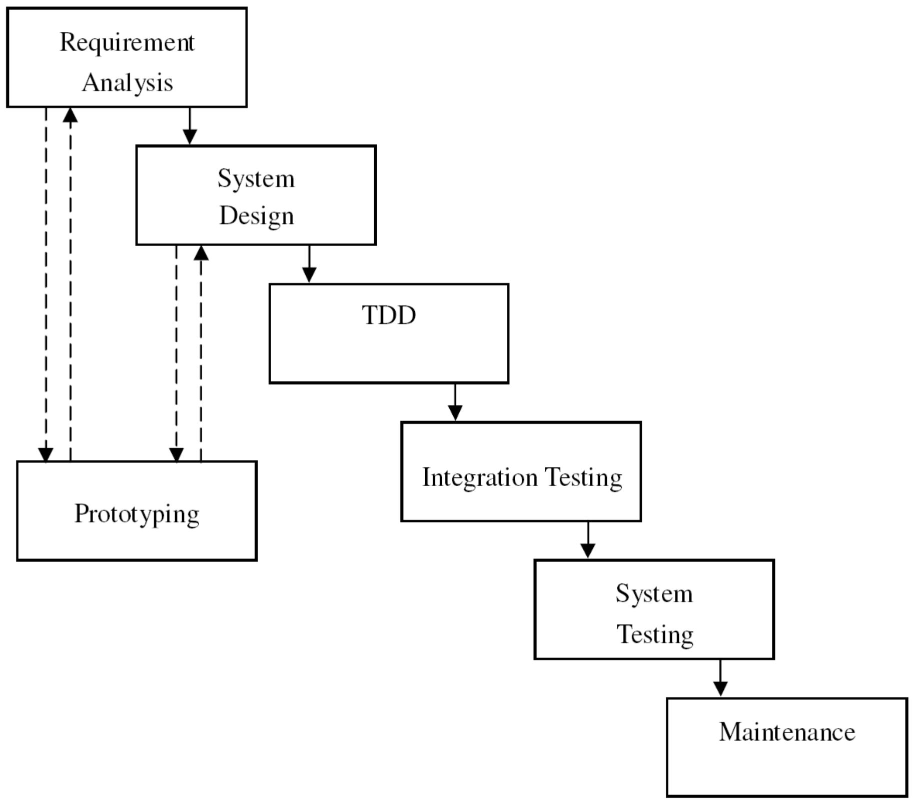

Figure 3. The waterfall model with prototyping and TDD

As illustrated in Figure 3, the TDD subprocess may be applied with many other process models. The new process model can be defined and tailored with TDD to meet the special needs of the project.

4. **Advantages of TDD.** The benefits of TDD are listed in the following.

(1) Iterative and incremental development. The iterative and incremental development is one of the most popular approaches in software development. TDD consists of a sequence of "red-green-refactor" cycles. It involves many iterations and allows developers to get feedback from previous iterations. From the constant feedback, developers feel confident about whether the new functionality has been implemented and whether the old functionality is still working.

(2) Better design. TDD not only validates the correctness of a component but also drives the design. Developers can make good design decisions because they cannot write

code without understanding exactly what the system behavior should be and how to test it. They are concerned with the interface of the component, not the implementation. This approach often leads to small and maintainable units with high level of cohesion. The frequent refactor in TDD also makes the code structure better.

(3) Easy debugging task. TDD takes small development tasks and relies on rapid response to small code changes in the form of executing the regression test suite after each change. The frequent nature of tests helps to detect bugs easily. Eliminating bugs early in software development reduces debugging time and avoids lengthy and tedious debugging later in the project. Repeated execution of tests gives developers a greater confidence in the code.

Although TDD has gained a lot of attention during the last decade, the adoption of TDD still faces many challenges. First, learning how to perform TDD may take several weeks, and lack of knowledge and experience with TDD may increase development time. However, as developers become familiar with TDD, the quality and productivity may be improved. Second, insufficient design is another concern, particularly in the development of large and complex systems. There are some studies reporting architectural problems in TDD. In our view, if we consider TDD as a subprocess, the problem with the lack of design in TDD can be solved. Developers must understand that TDD does not fit every situation. When developing a system, they must determine when to apply TDD and how to select an appropriate process and meld it with TDD.

5. Simple Experiment.

To investigate the effectiveness of TDD, we conducted a simple experiment with undergraduate students in the winter of 2015-2016. We selected 24 students who had finished object-oriented programming course and software engineering course. All these students were voluntary and interested in software development, and some of them had done several projects on object-oriented programming.

The students were assigned to six groups, each group with four students. Before the experiment we told them the goals of the experiment and gave them three days training on how to use TDD. Three groups applied the waterfall model with prototyping and used the traditional program design, coding and unit testing steps instead of TDD subprocess. The other three groups applied the waterfall model with prototyping and TDD. They followed the same software development process model except TDD subprocess.

The project we selected for the experiment was a simple application used for the student information management. Subjects were asked to perform the experiment task in the computer laboratory and follow the planned time schedule. In the experiment our aim was to find the effectiveness of TDD as a software development subprocess. After the completion of the project, subjects filled in questionnaires, we evaluated the systems they developed and examined the code. The data needed to measure the effectiveness of TDD were also collected. The major fault statistics are illustrated in Table 1.

Our analysis results are quite positive. Table 1 shows that all three TDD groups achieved better product quality than three non-TDD groups. They also created the systems which were less complex and more highly tested. Other design characteristics

Table 1. Major fault statistics

Group No.	Process Type	Major Fault Count
Group 1	Integrated TDD Process	10
Group 2	Integrated TDD Process	8
Group 3	Integrated TDD Process	13
Group 4	Traditional Process	22
Group 5	Traditional Process	15
Group 6	Traditional Process	20

such as coupling and cohesion were also improved. As to the productivity, we did not observe improvements for the three TDD groups. A possible explanation is that they needed to create more tests which increased the development time.

6. Conclusions.

Our research intended to investigate the effectiveness of TDD as a software development subprocess. Our results suggest that incorporating TDD into a software development process, some improvements in quality can be achieved. Developers can take advantage of the benefits of TDD in a software life cycle even if they have no experience in this technique.

The validity of our results could be limited because all participants in the experiment are students. Furthermore, the results are based on a small number of students. To generalize the results, our experiments should be repeated in different environments and in different contexts. We look forward to conducting the experiment in different contexts with industrial developers.

We would like to do further TDD research on larger projects and collect data for more accurate statistical analysis.

REFERENCES

[1] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Boston, MA, 2000.

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

[3] D. Astels, *Test Driven Development: A Practical Guide*, Prentice-Hall, 2003.

[4] D. Janzen and H. Saiedian, Test-driven development: Concepts, taxonomy, and future direction, *Computer*, vol.38, no.9, pp.43-50, 2005.

[5] J. Sanchez, L. Williams and E. Maximilien, On the sustained use of a test-driven development practice at IBM, *Proc. of AGILE*, pp.5-14, 2007.

[6] H. Erdogmus, M. Morisio and M. Torchiano, On the effectiveness of the test-first approach to programming, *IEEE Trans. Software Eng.*, vol.31, no.3, pp.226-237, 2005.

[7] L. Crispin, Driving software quality: How test-driven development impacts software quality, *IEEE Software*, vol.23, no.6, pp.70-71, 2006.

[8] N. Nagappan, E. M. Maximilien, T. Bhat and L. Williams, Realizing quality improvement through test driven development: Results and experiences of four industrial teams, *Empirical Software Eng.*, vol.13, no.3, pp.289-302, 2008.

[9] Y. Rafique and V. Misic, The effects of test-driven development on external quality and productivity: A meta-analysis, *IEEE Trans. Software Eng.*, vol.39, no.6, pp.835-856, 2013.

[10] E. Guerra, Designing a framework with test driven development: A journey, *IEEE Software*, vol.31, no.1, pp.9-14, 2014.

[11] S. L. Pfleeger and J. M. Atlee, *Software Engineering: Theory and Practice*, Prentice-Hall, 2010.