

Final report

by Hankun Chai

FILE	17863_HANKUN_CHAI_FINAL_REPORT_227501_414054811.DOCX (759.56K)		
TIME SUBMITTED	21-JAN-2020 12:14AM (UTC+0000)	WORD COUNT	3589
SUBMISSION ID	118809015	CHARACTER COUNT	19017

UWTSD ASSIGNMENT REPORT

Agile Software Engineering Principles

Hankun Chai
January 18th 2020

Abstract

This report introduces the basic characteristics and processes of TDD,BDD and FDD, compares their relative merits and differences, and finally gives the best possible application scenario.

Key Words: TDD, BDD, FDD, difference, scenario

Content

TASK 1

Abstract	2
1. TDD	5
1.1 Test Driven Development	5
1.2 Test-driven development cycle	5
1.3 Laws and Principles of TDD	6
2. BDD	7
2.1 Behaviour Driven Development	7
2.2 Behaviour Driven Development Cycle	8
2.3 Feature File	9
3. FDD	10
3.1 Feature Driven Development	10
3.2 How to use FDD	10
4 Relative merits of and differences between TDD, FDD and BDD	13
4.1 TDD and BDD	13
4.2 FDD	14
5 Reference	15

TASK 2

Abstract	18
1. High Cohesion and Loose Coupling	19
1.1 Definition of cohesion and coupling	19
1.2 Types of Cohesion and Coupling [1][2][3]	19
1.3 The impact of cohesion and coupling on software development	20
2. Information Hiding and Data Hiding	1
2.1 The definitions	1
2.1 The Role of Information Hiding and Data Hiding	1
Reference	3

1. TDD

1.1 Test Driven Development

Test-driven development(TDD) means that before we develop the code, we make the test cases, we test the code and find out what's missing, we complete the code as needed to just meet the requirements of the current phase, and then we loop through the code until we've completely solved the initial test cases. [1][2][3]

1.2 Test-driven development cycle

One of the main difference between TDD and traditional development is the procedures. For the traditional development, we usually do the coding after analyzing the requirement. And then we test the code, find and fix the bugs.

But for Test driven development, we develop our program as these procedures: [4]

- Analyzing the requirement and separating the concerns out.
- Instantiate the requirements and make every detail of the requirements clear.
- Write tests which only concerns the input and output of the program for the separated simple requirement.
- Write the code, regardless of other needs, and use the simplest way to meet the current small demand.
- Refactor the code to make the code clean.
- Repeat write-test → write-code → refactor-code until every demand which is separated from the requirements is met.
- Test and submit.

The basic process of TDD is: red, green, refactoring.

- Red: Write the test that fails
- Green: Make the code work

- Refactor: Eliminate redundancy

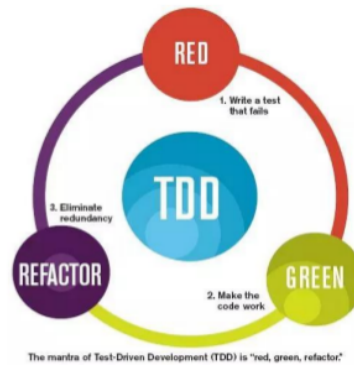


Figure 1: The Cycle of TDD

1.3 Laws and Principles of TDD

The 3 laws of TDD :

1. You can't write any production code until you have first written a failing unit test.
2. You can't write more of a unit test than is sufficient to fail, and not compiling is failing.
3. You can't write more production code than is sufficient to pass the currently failing unit test.[2]

After querying and reading some information about the 3 laws, I realized that what the laws were designed for.

For the first law, if you write the production code before your unit test, you will not know what the function of this code and you also can not make sure it does implement the required function.

If you break the 2nd law and write multiple failed tests, the test modification cost may be increased when refactoring.

If the production code implements a function that exceeds the current test, the excess code will not be protected by the test and required extra test.

It is said that the main purpose of the three laws of TDD is separating the concerns and focusing on one thing at a time.[5] There are several concerns during our programming: requirements, implementation, design. TDD gives us three clear steps, each step focusing on one aspect.

- Red: Write a failed test. It is a description of a small requirement. You only need to care about the input and output. At this time, you don't need to care about how to implement it.
- Green: Focus on implementing the current small demand in the fastest way, don't care about other requirements, and don't care how terrible the quality of the code is.
- Refactoring: There is no need to think about the requirements or the pressure of implementation. You just need to find out the bad smell in the code and eliminate it with a method to make the code clean.[4]

2. BDD

2.1 Behaviour Driven Development

BDD is a derivative of TDD, which describes and writes test cases from the perspective of a functional user in the same way that user stories or user use cases are written in a natural language or a quasi-natural language, through a specific BDD framework.[7][8][9]

2.2 Behaviour Driven Development Cycle

As shown in the picture, BDD is a supplement to TDD by writing FEATURE file.

The cycle of BDD is :

1. Business people and business analysts work together to discuss business requirements
2. Customers, business analysts, development testers, discuss specific business requirements together
3. Write structured Feature files that define the scenarios that will occur in many requirements and lists many specific examples
4. By these Featured files, the developer will know the detailed requirements, and turn them to unit test;

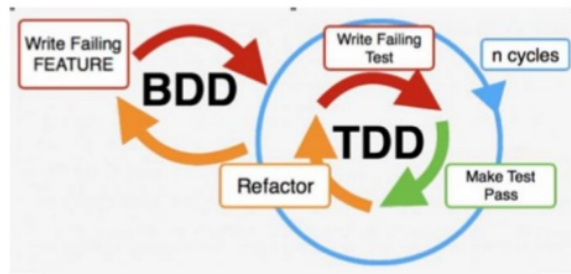


Figure 2 BDD Procedure

5. Using Test Driven Development to develop.
6. Complete and submit.

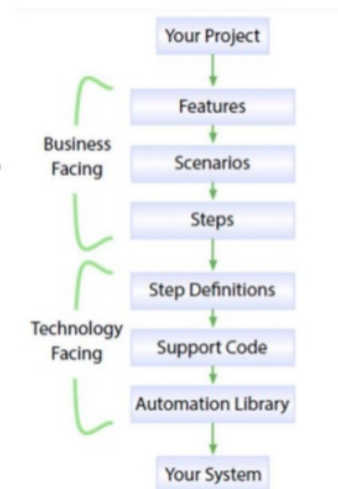


Figure 3 BDD feature steps

2.3 Feature File

The major difference between TDD and BDD is that BDD need every detail of every behaviour in user stories. And stored those scenarios and examples into feature files. BDD requires that business analysts and developers should collaborate, and that behaviour should be specified based on user descriptions, each explicitly documented in a dedicated document. [8]

Each user story should, in some way, follow the following structure:

```
Title: Returns and exchanges go to inventory.
"
As a store owner,
I want to add items back to inventory when they are returned or exchanged,
so that I can track inventory.
"
Scenario 1: Items returned for refund should be added to inventory.
Given that a customer previously bought a black sweater from me
and I have three black sweaters in inventory,
when they return the black sweater for a refund,
then I should have four black sweaters in inventory.
"
Scenario 2: Exchanged items should be returned to inventory.
Given that a customer previously bought a blue garment from me
and I have two blue garments in inventory
and three black garments in inventory,
when they exchange the blue garment for a black garment,
then I should have three blue garments in inventory
and two black garments in inventory.
```

BDD does not have any formal requirements for recording these user stories accurately, but it insists that every team that USES BDD come up with a simple, standardized format for recording user stories, including the elements listed above. [8]

3. FDD

3.1 Feature Driven Development

Feature driven development (FDD), one of the agile development methodologies, was developed by Jeff DE Luca, Eric Lefebvre, and Peter Coad in collaboration.[10]

He suggests that each feature should be developed in a short period of time, say 2 weeks, that it should be executable, and that the development of the project should be monitored accurately and timely. [10]

F-D-D also allow the business analysts to work with the developer and participate directly in the system building. [10]

3.2 How to use FDD

3.2.1 Feature

In FDD, Feature is a basic development unit, an increment in a (FDD) project, and refers to the smallest useful function in the eyes of users, which can be implemented in a very short time (usually within two weeks).[11]

- The features are small

Features are small because they can be implemented in two weeks, and any function that should be ensured to be completed in about 2 weeks can be further decomposed into features small enough to be called a feature. Making features smaller also means that customers often see measurable progress.

- Features are of customer value

In a business system, a feature maps to a step of some activity in a business process. In other systems, the feature is equivalent to a step in a task performed by the user.

3.2.2 Feature Driven Development Procedures

FDD approach consists of five processes:

- Develop Overall model[10]
- The initial effort of a FDD project is to gather the functional requirements of the system and then model them globally. Generally, we use a four-primary color model
- Build feature list
This process identifies all the functions required by the requirements. Refine the project into a number of operations or activities based on the work of the domain expert. Then convert each step into a function. The result is a list of functions at each level
- Plan by feature
After the feature list is completed, the next stage is to generate the development plan and assign ownership of features to programmers as classes.[10]
- Design by feature
Period specified by the project manager and each feature set together with the main programmer to carry on the detailed design the function and the domain model on the basis of analysis and design. [10]
- Build by feature
Code implementations by design, with the programmer implementing the respective classes. Organize code review and review as necessary after code completion. After the tests and checks pass, check in to the configuration management library to build. [11]

Build by feature and Design by feature is an iterative process. In this way, functions can be realized continuously through continuous iteration.

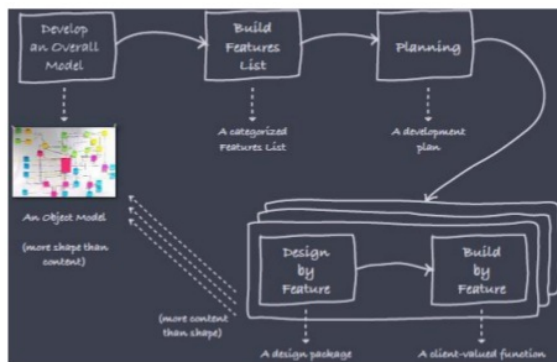


Figure 4 FDD Process

4 Relative merits and differences between TDD, FDD and BDD

4.1 TDD and BDD

Like TDD, BDD is also a test-first approach, but differs by testing the actual behaviour of the system from the end users perspective.[12]

The shared merits are mainly from test-first. Such as:

- By obeying the three laws, we can easily control our concerns and focus on one thing so that we can reduce the probability of error and save our energy.
- By writing small test at a time, the code can be more modular. And it also be helpful with understanding and reading when the code need to be modified.
- It helps to clarify requirements because you have to figure out concretely what inputs you have to feed and what outputs you expect.[6]
- Because we write test before we write code, it will not be necessary to test again.

And for BDD, the main advantage is that BDD also allows non-technical people to participate in the discussion of developing, and engineers can take the specification and turn it into a test case to make the specification an "executable specification," making it easier for people from different backgrounds to agree on the software project.

Comparing with TDD, BDD can reduce mistakes in selecting and completing features.

BDD focuses on providing specific functionality rather than the smallest possible unit while TDD is dedicated to testing smaller features in isolation

The feature file of BDD also contains many examples which are helpful when writing the code.

TDD may be suitable for those projects whose requirements are entrenched. The test cases of TDD are written with the test unit, and they are hard to change when the requirements are changed.

The best-applied scenario could be larger project which has the complex design and requires good communication between the customers and development team. Because of the existence of the feature file, the requirements to development team can be more understandable and accurate. And the cost of analyzing requirements and TDD is too high for a small project or a small development team.

4.2 FDD

The biggest difference between FDD and BDD/TDD is that FDD does not use the method which is the cycle :Test_fail→WriteCodeToPass→Refactoring.

Comparing with BDD or TDD, developer in TDD would write an overall model which will be useful in many cases like estimated cost.

Features in FDD are designed as a task which can be implemented in a very short time (usually within two weeks). That will help to bring work done in a short time and easiest manner[13].

Like BDD, FDD requires that customer and business analyst should join to the project.

So the best application of FDD could be moving larger size project. Because these existed project already have their overall model, detailed requirement.

Another scenario could be a large size project which needs complex system. Because FDD can break the entire problem into many tiny ones.

5 Reference

- [1] 'Why does Kent Beck refer to the "rediscovery" of test-driven development? What's the history of test-driven development before Kent Beck's rediscovery? - Quora'. [Online]. Available: <https://www.quora.com/Why-does-Kent-Beck-refer-to-the-rediscovery-of-test-driven-development-Whats-the-history-of-test-driven-development-before-Kent-Becks-rediscovery>. [Accessed: 17-Jan-2020].
- [2] Beck, Kent. *Test-driven development: by example*. Addison-Wesley Professional, 2003.
- [3] 'Test-driven development', *Wikipedia*. 16-Jan-2020.
- [4] '深度解读 - TDD（测试驱动开发）'，简书. [Online]. Available: <https://www.jianshu.com/p/62f16cd4fef3>. [Accessed: 17-Jan-2020].
- [5] 'The 3 Laws of TDD: Focus on One Thing at a Time • Quality Coding', *Quality Coding*, 23-Jan-2018. [Online]. Available: <https://qualitycoding.org/3-laws-tdd/>. [Accessed: 17-Jan-2020].
- [6] 'Pros and Cons of Test Driven Development', *Testing Excellence*, 29-Nov-2016. [Online]. Available: <https://www.testingexcellence.com/pros-cons-test-driven-development/>. [Accessed: 18-Jan-2020].
- [7] 'Introducing BDD | Dan North & Associates'. [Online]. Available: <https://dannorth.net/introducing-bdd/>. [Accessed: 18-Jan-2020].
- [8] 'BehaviourDrivenDevelopment - BDD Wiki', 01-Sep-2015. [Online]. Available: <https://web.archive.org/web/20150901151029/http://behaviourdriven.org/>. [Accessed: 18-Jan-2020].
- [9] 'Behavior-driven development', *Wikipedia*. 07-Jan-2020.
- [10] 'Feature-driven development', *Wikipedia*. 12-Sep-2019.

[11] ‘敏捷开发系列之旅 第三站（认识 FDD 特征驱动开发）_紫羽风的博客-CSDN 博客’. [Online]. Available:

<https://blog.csdn.net/happylee6688/article/details/22372067>. [Accessed: 19-Jan-2020].

[12] ‘TDD vs BDD - What’s the Difference Between TDD and BDD?’, *TestLodge Blog*, 18-Apr-2018. [Online]. Available: <https://blog.testlodge.com/tdd-vs-bdd/>.

[Accessed: 19-Jan-2020].

[13]‘Top 12 Software Development Methodologies & its Advantages / Disadvantages | TatvaSoft’. [Online]. Available: <https://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-disadvantages/>.

[Accessed: 19-Jan-2020].

UWTSD ASSIGNMENT REPORT

Programming and Software Testing

Hankun Chai
January 18th 2020

Abstract

This report introduce the definition of coupling, cohesion, information hiding and data hiding. Then discuss the importance of low coupling and high cohesion of software development. At last, illustrate the how the information hiding and data hiding. secure the data and program.

Keywords: coupling, cohesion, information hiding, data hiding

1. High Cohesion and Loose Coupling

1.1 Definition of cohesion and coupling

Cohesion of a module shows the degree to the function of the extent of relevant procedures combined into one module , or function of the degree of condensed state or degree [1].

The closer the relationship between modules, the higher the cohesion, the more cohesive it is.

Coupling is a software metric that refers to the degree to information or parameter dependence between modules and between modules in a program.

The complexity of the interface, the method of invocation and the information passed determines how much is the coupling between modules.

Cohesion is always contrasted with coupling in many occasions. High cohesion always occurs with low coupling, and vice versa.

1.2 Types of Cohesion and Coupling [1][2][3]

Coupling:

- Content coupling : the content of one module could be directly accessed by another module. And this type has highest coupling.
- Common coupling means that several modules all can access the same global data environment.
- External coupling means that two modules all access the global simple variables.
- Control coupling: The transfer between modules is control information such as flags. The functions of a module are controlled by another one.
- Stamp Coupling means that it is data structures passing from the calling module to the called one. Tables and modules deliver data results such as data names, which are tags that pass addresses.
- Data coupling means only simple data item parameters can be passed from the called module to the calling one.
- Indirect coupling: there is no direct relationship between the two modules, and the connection between them is relied on the main module.

So the strongest coupling is common coupling, at this situation, whether this module can work properly depends entirely on the correctness and normal operation of the other module.

And the degree of coupling is weakest in indirect coupling. There is no direct connection between these modules.

Cohesion:

- Coincidence cohesion means that the parts within a module are not related to each other, but are accidentally brought together. This type has lowest cohesion.
- Logical cohesion means which function should be performed is determined by the parameters which are passed to it when the module is called.
- Temporal cohesion: modules that combine operations that need to be executed simultaneously are called temporal-cohesive modules.
- Procedural cohesion: An operation is designed to be activated immediately after the completion of another operation.
- Communicational cohesion: It means that all processing elements in a module operate on the same data structure.
- Sequential cohesion: each operation in a module serves the same function and must be done sequentially, usually input of the second operation element is the output of the first.
- Functional cohesion means each component of all elements in the module exists for the purpose of accomplishing the same function, which is a single function, and the module is no longer separable.

So the weakest cohesion is coincidental cohesion. And the highest cohesion is the functional cohesion, every element inside the module is designed for cooperation.

1.3 The impact of cohesion and coupling on software development

The coupling between modules represents the degree of dependence on of a module against another. If proper functioning of a module A in a system highly depends on another module B, which means high coupling, module A will not work when B is in problem like being replaced or re-designed. In this kind of situation, it will cost much to make the system work again. Module A should be re-designed and any other

module which has strong coupling with A need to be modified too. If the coupling between most of modules of the whole system is high, the system should be totally re-designed once a module goes wrong.

Low cohesion shows the high degree of isolation between operations inside a module.

It will

- Make it hard to understand the module.
- Increases the difficulty of maintaining the system because a logical change affects many modules, and a change to one module causes some related modules to be modified.
- Increases the difficulty of module reuse because most applications cannot reuse a module composed of many functions that are not necessarily related.

In most of circumstances, high cohesion means high readability and reusability, and a system with lower coupling has better reusability, expansibility and maintainability, which would benefits the development and maintenance of the system.

However, all we said does not mean higher cohesion is better, lower coupling is better. For instance, the strongest form of cohesion is when a class writes only one function, which is the most cohesive. But this will bring one.

The obvious problem: the number of classes increases so dramatically that other classes are coupled so much that the whole design becomes "highly cohesive and highly coupled." Because of the high coupling, the whole system also changes very frequently.

2. Information Hiding and Data Hiding

2.1 The definitions

The concept of data hiding always is regarded as encapsulation. Its definition is to combine the properties and operations of the object into a single entity and hide the implementation details inside the object as much as possible. Visibility of properties should also be determined when they are defined, that is, which features are not visible to the outside and which features are, those properties can only be invoked by internal functions.[5] Access to the data is controlled only through defined interfaces, controlling the extent to which the user modifies the class and accesses the data.

Information hiding is known as base of object-oriented and structured design. [7]

The definition of Information hiding is that modules which do not need the specific information (process or data) in another module when designing cannot access the information. That is, the implementation details in one module are not publicly available. only simple interfaces are designed left for other modules to access. A developer writing a program does not have to waste a lot of time understanding how to implement functionality within its module, and the module is called to do the job.

2.1 The Role of Information Hiding and Data Hiding.

Data hiding:

- After encapsulated object (the object here is refers to the code of program unit, generally refers to: assembly, namespaces, classes, methods, properties, variables, etc.) the external object is unable to direct access to the object, the internal implementation details of internal implementation details of the changes will not affect the external object access principle, that is a reflection of isolation. And it also improves cohesion and reduces coupling.
- The encapsulated object can be accessed by multiple external objects without specifying different service objects for each external object. [6]
- The name of the encapsulated object, if properly named, can be understood without looking at the implementation details.[6]

Information Hiding:

- Hidden complexity: so you don't have to deal with it anymore, unless you're paying special attention;
- Hiding the source of change: so that when a change occurs, its impact can be localized. The root causes of complexity include complex data types, file structures, Boolean judgments, and obscure algorithms.[4]
- Information hiding also helps to expose interfaces to design classes. At all levels of design, you can make good design decisions by asking what to hide[4]

Reference

- [1] ‘面向对象原则：高内聚、低耦合。多聚合、少继承_面向对象,高内聚,低耦合’. [Online]. Available: <https://blog.csdn.net/kingscoming/article/details/78836229>. [Accessed: 20-Jan-2020].
- [2] ‘Cohesion (computer science)’, *Wikipedia*. 18-Dec-2019.
- [3] ‘Coupling (computer programming)’, *Wikipedia*. 13-Jan-2020.
- [4] ‘Java 学习笔记之信息隐藏(Information Hiding)_贱贱的梦想-CSDN 博客’. [Online]. Available: <https://blog.csdn.net/GongchuangSu/article/details/53895916>. [Accessed: 20-Jan-2020].
- [5] ‘What is data hiding in a program? - Quora’. [Online]. Available: <https://www.quora.com/What-is-data-hiding-in-a-program>. [Accessed: 20-Jan-2020].
- [6] ‘深入分析面向对象中的封装作用 - 梦在旅途 - 博客园’. [Online]. Available: <https://www.cnblogs.com/zuowj/p/5427291.html>. [Accessed: 20-Jan-2020].
- [7] ‘信息隐藏_百度百科’. [Online]. Available: <https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E9%9A%90%E8%97%8E>. [Accessed: 20-Jan-2020].

Final report

GRADEMARK REPORT

FINAL GRADE

81 / 100

GENERAL COMMENTS

Instructor

Task 1 report (35%). 30

Task 1 use of git (15%) 12

Task 2 report (20%) 14

Task 2 code implementation (30%) 25

Comments

=====

The discussion of TDD and FDD is relevant and interesting. A few more examples could have been provided to illustrate their application. More information should have been provided on the practical implications of coupling and cohesion in a software engineering context.

Code works and implements all thre required features.

Good usage of GIT.

Software tests not provided.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24
