

## Abstract

This is an article about Test Driven Development (TDD) and Test Last Development (TLD), and how they differ, and when to use one over the other.

## 1 Introduction

The objective of this article is to research the best approach to test code during development. The two different testing strategies covered in this article will be test driven development (TDD) and test last development (TLD). The article will try and explain the difference between the two, through the perspective of multiple articles. Then conclude on which strategy is the most efficient way to probably test your application code.

The article is build up of four sections:

- **Problem definition 1.1:**
- **Hypothesis 2:**
- **Analysis 3:**
- **Conclusion 4:**

### 1.1 Problem definition

It is easy to develop an application from a list of specifications, but it is a lot harder to make it bug free and stable. To help avoid these problems, developers use the strong tool of testing code. But how should a program be tested. This article will look at two different approaches:

- Test Driven Development (TDD)
- Last Test Development (LTD)

And find the answer to which of the two strategies is best applicable to development.

## 2 Hypothesis

The hypothesis is that TDD is the superior testing strategy compared to TLD.

The general philosophy in many articles and teaching institutes is that TDD is the right way to go when testing code, compared to other strategies like TLD. The hypothesis is that TDD is way more effective at covering all the code, as the developer programs the test on the way to the finished product. Where TLD is very time consuming, it is often placed at the end of the development cycle. This makes it harder to account for the time needed for a full system testing. It

is easier to forget certain functions and can result in a lot of technical debts. In the TDD approach, the developer team account for the testing when developing the function.

### 3 Analysis

The analysis will be composed from a range of articles and write-ups by authors who argue for and against TDD or TLD. Some will also be more objective views and focus more on the comparison of the two, and when one excels over the other. Some of the articles will be subjective with the purpose of getting you to subscribe to one strategy. Then build a conclusion from their citations and experience.

#### 3.1 Why TDD?

Code becomes modular: When using TDD, the code becomes more modular. The reason for this is that a developer writes a test and then writes the code to satisfy the test. This makes the developer think more in modules as they are easily tested. The real downside here, is that the developer is going to use more time writing the code.

Detect bugs early: When writing the test first, the developers get instant feedback if the code is working as it should, by simply running the test. This will help with finding and fixing bugs earlier.

Cheaper to fix: When a developer can get instant results on whether the code has passed or not, the developer will quickly change the code to be correct, while his mind is focused on this piece of code. If the developer comes back to test the code even a few days later, he will first have to get the code into his head, which takes a lot more time. A drawback can be, that the test suites becomes too big and therefore takes too long to execute. Then there is a lot of time needed to be invested into reducing the test suite.

Refactoring becomes easier: When a developer has well defined tests in place, refactoring code is remarkably simple, as the developer would get instant results on whether this refactored code has passed or not. Writing a well-defined test however, can take a long time, and in the most extremes, the tests for the code can take more time than writing the code itself.

Faster development: When a developer is writing some code and want to test it, if they are not using TDD, they would have to manually do the tests. This could take a lot of time. But if there are automatic tests, this can be done in a fraction of the time. If, however functionality of the code is changed rapidly or functions are added just to be removed later, the development would suffer. Because test suites for the function would still have to be made first.

Cleaner Code: Starting to use TDD can be hard, because it changes the way of thinking when it comes to writing code. This will be hard and take a lot of effort. Overtime however, the more experience the developer gets, the easier it

becomes, and the code would be better and more clearer on the first pass at writing the code.

### 3.2 Why TLD?

When looking at the TLD strategy, it can be hard to find a specific “pro TLD” approach in the articles. This comes down to the fact that testing will always be important in the development cycle. This could be the result of TLD already functions well. But now TDD has entered the space of development, and challenges the normality of testing. Because of this, the analysis will look more at the arguments against TDD and turn them around to see if they have a positive counter argument in TLD.

In the medium article by Simon Redmond [2] he list a few arguments as to why he believes that TDD is not as smart and quick as it is often claimed to be, and how it may accidently trap someone in an unhealthy development pattern.

Redmond’s main argument to the flaw of TDD is that, “You end up writing your code to satisfy your test rather than writing code to build your product” [2]. In the article he presents a piece of code, for grading papers (Listing 1).

Listing 1: Grading code snippet

```
1      public char GetLetterGrade(int grade)
2      {
3          if (grade > 90){return 'A';}
4          if (grade < 90 && grade > 80){return 'B';}
5          if (grade < 80 && grade > 70){return 'C';}
6          if (grade < 70 && grade > 60){return 'D';}
7          if (grade < 60){return 'F';}
8          return 'F';
9      }
```

The code has the obvious bug that, if a person gets an exact score of 90, the code returns an “F”. He argues, that a competent developer should be able to see the mistake beforehand, and going through a long process of writing test before writing the actual method would be an unnecessary time cost. If the developer use TDD or TLD, the bug would have been caught eventually.

Redmond’s concern is not about TDD’s ability to write test or code, but if it adds any actual value. Redmond’s biggest concern is that, if developers write code to satisfy a test, they will lose focus on the actual product, that it takes away the common sense from the developer and ultimately stop the developer from being innovative.

To sum up Redmond’s article. TDD is not a failed test strategy, but there is a lot of extra complexity that might not add any value to the overall process. While bugs get noticed earlier in TDD, they will eventually be caught in TLD also. In TLD however, might have less complexity, it does not take a huge

amount to write tests, to then satisfy them. When the developer approach testing at the end of the development (TLD) the developer also get a more complete picture of what they are testing, and can write well defined tests a lot easier. The process is a lot easier to understand, allowing for more creative powers to play in the actual development.

It is worth noting that Redmond's article is subjective, and does little to try actually argue his points, more so than stating them.

### 3.3 TDD vs TLD - Pros and cons

Each of the two test strategies has their pros and cons. Some of the most common pros and cons can be seen in the table 1 and 2

| TDD                         |  |
|-----------------------------|--|
| Pros                        | Cons   |
| Code becomes modular        | Takes a lot of extra coding                    |
| Decect bugs early           | Can result in major changes to the code later  |
| Cheaper to fix              | Require often reconfiguration of suites        |
| Refactorring becomes easier | Hard to make well defined tests                |
| Faster development          | A lot of wasted time in constant changing code |
| Cleaner code                | Steep learning curve                           |

Table 1:

| TLD                                 |   |
|-------------------------------------|---|
| Pros                                | Cons                                    |
| Easy to do and understand           | Expensive to maintaine                  |
| Code is ls less complex             | Testing can become too wide             |
| Does not require extra refractoring | Bugs are discovered late in development |
| Has a lot less code                 | No pre-testing of new functions         |
|                                     | Risk of missing new bugs                |

Table 2:

## 4 Conclusion

It is hard to say which method is the best for testing. According to Susan Hammond [1], it is naïve to claim that TDD is better, because it is not simple to define or measure if something is TDD.

The conclusion is therefore not going to point to TDD or to TLD as the best way to go. Ideally it is up to every individual team to decide from project to project, if they should use TDD or TLD for this project. Understandably this will not really happen, the more realistic scenario will be, that the team will stick to one method, and use that exclusively. This might overtime prove just as good, considering that the team will become a lot better to that testing method.

## References

- [1] Susan Hammond. Risk/reward analysis of test-driven development. <https://etd.auburn.edu/bitstream/handle/10415/6890/Susan%20Hammond%20Dissertation.pdf?sequence=2&isAllowed=y>.
- [2] Sam Redmond. Why we shouldn't use tdd. <https://medium.com/@sredmond/why-we-shouldnt-use-tdd-fcd12992d093>.