Written: 11-12-2020 - Revised: 16-12-2020

Test Driven Development or Test Last Development

Written by

Alex Langhoff & Andreas Due Jørgensen

Abstract

Is Test Driven Development (TDD) better then Test Last Development(TLD)? Choosing the wrong test strategi would be costly and could end up in projects that dont add value to your company. Even if alot of sources says TDD is always the right choice, it really isnt. There is no simple way to determine or meassure if one is better then the other, its is important for a development team to choose the right strategi for the project at hand.

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.

1.1 Problem definition

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

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

Then conclude which of the two strategies is best suited the assignment at hand.

1.2 Constraints

There will be a few constraints that has been taken into consideration. TDD and TLD is far from the only methods of testing during code development. However this article will be limited to the differences between TDD and TLD.

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 developerteam 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 fiding 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 to big and therefor 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 TLD

TLD is a very different from TDD. Test Last development does not mean to test at the end of the total application. But it entails that the developer execute some unite tests against the code when the developed unit is done. All in all does it mean that all testing is done before code goes into Production. But not automated. [4]

TLD Pros and Cons It takes less time than TDD. TDD takes up to 16% longer than TLD. [1] This is due of the high learning curve of TDD. A lot of time in TDD is used on skipping between code and tests and focusing on writing failing tests. Using TLD the code base becomes a lot smaller, and more linear and easy to understand, with simpler code and less Cylomatic complexity. The developer has to spend less time refactoring as the code is written to solve the problem and move on to the next specification. Some of the things TLD lacks is that it has up to 52% less test than TDD, due to TDD's aggressive testing. This leads to more bugs in a TLD system. The code also tent to become "uglier" and more solidified, making it hard to change later. Lastly, time is money, and TLD requires a lot more time and effort to maintain later. [6]

When skimming the pros and cons, we get the idea that, while TLD is easier, the quality is inferior to TDD. Higher quality code is one of the pros of TDD often mentioned. In 2005 by Lech Madeyski made a study on the correlation between TDD, TLD and code quality. [3]. The study has 108 developers, spread across experience and coding language. They experiment runs over eight coding sessions, the participants are divided into 4 groups.

- Solo programming TLD
- Solo programming TDD
- Pair programming TLD
- Pair programming TDD

After the eight sessions, the code is inspected and graded on quality. Lech Madeyski found that the code quality (Measured by acceptance tests passed) was highly affected on which approach was used. What is interesting though was that he found that Solo programmers who used TLD had a better code quality that Solo programmers using TDD. He also found that there was no difference in code quality from the Pair programming groups. This gives a picture, that it is cheaper and better to have a single developer during TLD, than two developer using TDD. [3].

The medium article by Simon Redmond [5], is a more anecdotal and less scientific source, echoes part of Lech Madeyski's conclusion. Redmon believes that using TDD over TLD is a costly effort for very little value. And talks about how aggressive testing for bugs, and writing code to satisfy code. Is time taken away from innovative code and the actual code that progress your product. Possible bugs and errors will be handled later when TLD, just like they would be caught in TDD. Something that can be recognized in the experiment.

Each strategy has its strength and weakness, but the learning curve of the two approaches is one of the big factors when choosing. If you have a more experienced developer team, that has the discipline and overview to write a test first, then write the method. Then the team should consider TDD over TLD. If the developer team has less experience, and are more of the "I need this to work now" kind of developer, the TLD approach should be considered.

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 [2], 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 shouldgit 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] Boby George and Laurie Williams. A structured experiment of test-driven development. https://www.researchgate.net/publication/222831281_A_structured_experiment_of_test-driven_development.

[2] 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.

[3] Lech Madeyski. Software engineering: Evolution and emerging technologies. https://books.google.dk/books?hl=da&lr=&id=dv7uAgAAQBAJ&oi=fnd&pg=PA113&dq=Test+driven+development+statistics&ots=HvxgrCZgJr&sig=iWRjHyvkdUFSIoJJFkXacvms7TQ&redir_esc=y#v=onepage&q&f=false.

[4] Gerard Meszaros. test-last development. http://xunitpatterns.com/test%20last%20development.html.

[5] Sam Redmond. Why we shouldn't use tdd. https://medium.com/@sredmond/why-we-shouldnt-use-tdd-fcd12992d093.

[6] Santosh Shah. Test driven development (tdd) vs test last development (tld): A comparative study. http://compilehorrors.com/test-driven-development-tdd-vs-test-last-development-tld-a-comparative-study/.