Experiment on Performance Difference between Test Driven Development and Code First Test After

Mads Riisom
University of Southern Denmark
Odense, Denmark
marii13@student.sdu.dk

Tenna Cortz
University of Southern Denmark
Odense, Denmark
tecor13@student.sdu.dk

Henrik Bolding Frank
University of Southern Denmark
Odense, Denmark
hefra13@student.sdu.dk

ABSTRACT

Test Driven Development (TDD) is considered a good approach concerning productivity compared to the more common Code First Test After (CFTA). The group aim to investigate if there is a performance gain in using Test-Driven Development (TDD) over Code First Test After (CFTA). The group performed an experiment using 65 test subjects, who received one of four coding tasks, the tasks had the same end result, but the tasks were described and had to be solved in different ways. The experiment showed that with this test group the approach was less important than how detailed the task was described. From this experiment we cannot make any generalized conclusion as the test group is not a representative of the target group, but there seem to be no real difference between TDD and CFTA.

Keywords

Test Driven Development, Code First Test After, Performance, Development Approach, Software, Engineering, Experiment

1. INTRODUCTION

In order to develop software faster, several approaches are available. The goal of this experiment, is to determine whether Test Driven Development, or Code First Test After approach has an increase in productivity, when developing software. In TDD the unit tests are written before the actual production code. When the test case has been described, just enough code to make the test case pass, is developed. In CFTA code is tested, as the production code is being developed.

1.1 Reading guide

First the research method will be described, in order to establish a base terminology to the reader. This section will also describe the requirements to the experiment as well. A conclusion will be made, by the result presented later in this article. Also threads to validity will be considered, when drawing this conclusion.

DOI: 10.475/123_4

2. RELATED WORK

While TDD and CTFA is not newly developed approaches, proper studies in this topic has already been made. As a part of this experiment, a research kit has been developed. This kit has been developed by D. Fucci, B. Turhan and M. Oivo, which also have conducted an experiment with the same goal. These studies show positive signs of increase in productivity, when using TDD [1].

3. RESEARCH METHOD

The experiment aimed to investigate the performance difference between TDD and the CFTA approach, with either step by step instructions (Sliced) or free text (Non-sliced). To investigate this we performed an experiment using third semester Software Engineering bachelor students at SDU as test subjects.

Before and after the experiment was conducted, the students were handed a pre- and a postquestionnaire respectively. The prequestionnaire contained questions regarding the students perception of their own skill level, where the postquestionnaire contained questions about how well the students thought they had performed during the experiment.

The task the students were to develop were based on four possible tasks that were distributed among the students. They were to develop an application that could calculate the bowling score of a single game, without a graphical user interface. Further the students had to develop a software solution and provide a test environment, that tested the requirements of the task. These requirements for the calculations were explained in the task descriptions, where it were expected for the students to use the specified approach. The tasks were as follows:

- Non-sliced, test-last (NSTL)
- Sliced, test last (SLTL)
- Non-sliced test first (NSTF)
- Sliced, test first (SLTF)

The test-last approach tasks were where the students had to implement the requirement, followed by implementing the test which ensures the fulfillment of the requirement. The test-first approach tasks were where the students had to implement a test to fulfiill the requirement, and then implement as little code to satisfy the test. The sliced tasks

Table 1: States

States	Description
Failed state	Solutions that has no chance to run in a centralized environment
Critical state	Solutions that has a high probability to run
Passed state	Solutions (partically) fulfilling the requirement

were where the requirements were described step-by-step, whereas the non-sliced tasks had requirement descriptions in one full textblock.

The four tasks were randomly distributed among the students with no knowledge to us of which students got which specific task. This were decided as the group had previous knowledge of the students strengths and weaknesses, and could therefore insinuate a biased distribution.

While performing the experiment the group participated in a supporting capacity, as well as observers and coordinators. While helping the students with the difficult parts of the experiment, the group gained more insights into the different tasks, and the students' approaches to the tasks. These insights could influence the group's objectivity to the experiment process as well as their perception of the results.

At the beginning of the experiment the students received a code skeleton to work from, to ensure that the solutions would have a similar structure, and therefore would be easier to compare. The submitted solutions was evaluated using an ordinal scale, from where it were decided whether the student were in one of the following three states; failed, critical and passed. These are described in table 1.

Afterwards the solutions' states and questionnaires were analysed to see if there was a correlation between the test-first or the test-after approach, and the task description's granularity.

4. RESULTS

The dataset the experiment is based on, consists of 65 data points, from where 3 data points were rejected as they did not fulfill the requirements of the questionnaires, in other words they were either double or incomplete submission. Therefore, only 62 data points are represented in the following section.

All the subjects had submitted a zip-file containing their solution based on the code skeleton provided for the experiment. The 3 states the submissions were categorized in, were evaluated as if it was a real exam submission. The states were divided as follows:

- 23 - failed state attempts
- 16 - critical state attempts
- 23 - passed state attempt

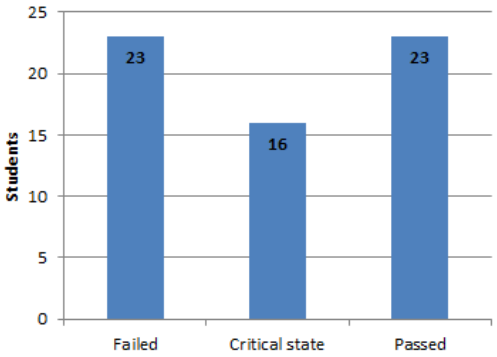

Figure 1: Distribution of student's performance

Figure 1 represents the distribution of student's performance in the experiment, where the evaluation of their submitted code were divided into the three states.

As the distribution of the tasks were random, and each student got exactly one task. The number of students receiving the individual tasks were distributed as illustrated in figure 2:

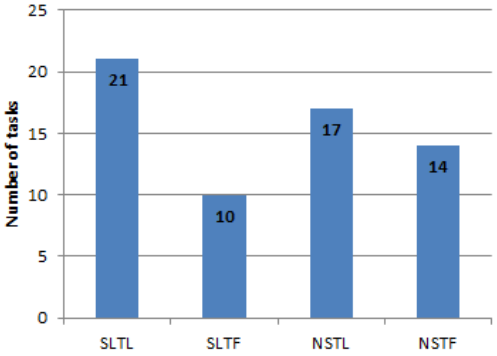

Figure 2: Task distributions

38 students used the test-last approach, and 24 students used the test-first approach. Further it is an exceptional case where both the sliced and non-sliced tasks were equally distributed with 31 of each.

After the assessment of the code submissions, it is possible to look at the answers of the questionnaires. Shown in figure 3 the student's perception of their own programming experience in these fields; General programming, C#, Visual Studio, Unit testing, NUnit testing framework and Test-driven development (TDD). The students were rating their experience on the scale: None, Novice, Intermediate and Expert.

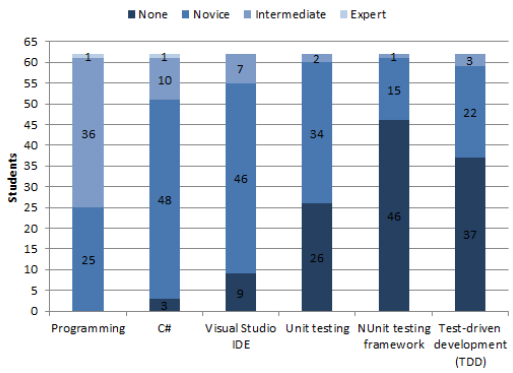

Figure 3: Student's own experience rating

It is interesting to see that the students who ended up with TF tasks rated themselves lower in their perception of their experience in TTD.

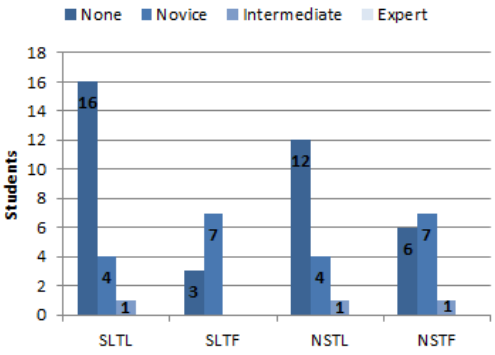

Figure 4: TDD experience in tasks

When looking at figure 3 and 4, there are more people with none TDD-experience in the test-last approach, which also is an exceptional case.

The experiment aimed to investigate the performance difference between Test-Driven Development (TDD) and the more common Code First Test After (CFTA) approach, with either step by step instructions (Sliced) or free text (Non-sliced).

As the experiment aimed to investigate the performance based on the four tasks distributed amongst the students, figure 5 shows the code states divided by the tasks.

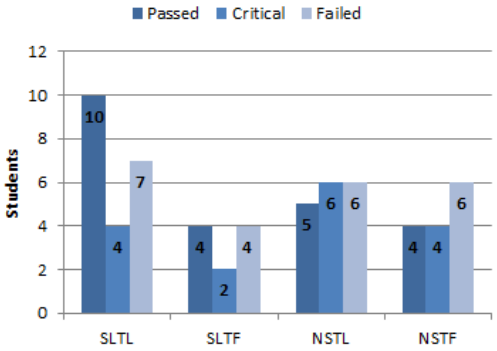

Figure 5: Student's task performance

As it is stated in figure 5, there are a tendency for the students' submissions to pass with sliced tasks.

Figure 6 shows the distributions between the students self-rating of programming experience level (None, Novice, Intermediate, Expert), and their correlating submission state (Passed, Critical, Failed), divided into each of the tasks respectively. It is important to notice that there were no students who have rated their programming experience as "None" which means that they could have excluded this in figure 6-9. These are however included in the diagrams as it was an option in the pre-questionnaire, and therefore if excluded, could give a skewed view of the students programming experience levels. This does also apply for the "Expert" ratings which is described show in figure 6.

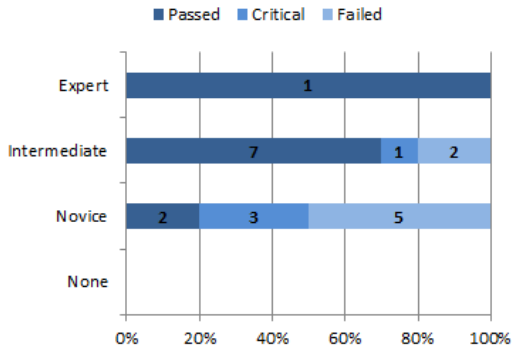

Figure 6: SLTL percentage distributions with relative task numbers

Here it is seen that 100% of all students who rated their programming experience to be "Expert" has submitted a solution that fulfilled the requirements satisfying, where it was only 70% and 20% in respectively "Intermediate" and "Novice". It is important to notice that there were only one student who rated him- or herself as an "Expert" in programming experience, which means that he could have been excluded from the statistical analysis of this task, as the "Expert" percentage are misleading the reader. The rest of the percentages are for the critical submissions 10% and 30%, and for the failed submissions 20% and 50% respectively for the "Intermediate" and the "Novice" students.

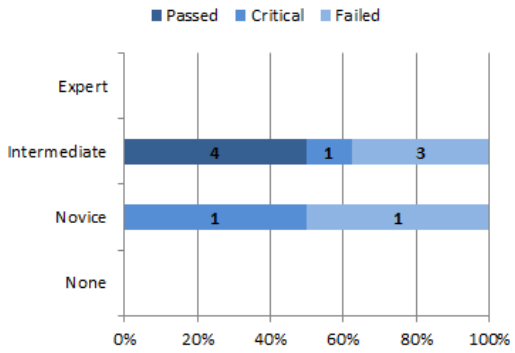

Figure 7: SLTF percentage distributions with relative task numbers

In figure 7 it is seen that only 50% of the "Intermediate" students submitted solutions which fulfilled the requirements satisfying, where there are none "Novice" students who did. On the other hand it is seen that it is respectively 12.5% and 50% in the "Intermediate" and "Novice" who were in the Critical state. The last percentages are 37.5% and 50% for respectively the "Intermediate" and the "Novice" students are those who failed.

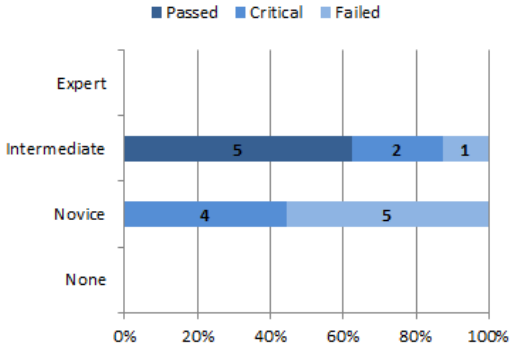

Figure 8: NSTL percentage distributions with relative task numbers

Figure 8 shows the distribution of the students' experience and states of the NSTL tasks. Here it is also seen that it is only the students who have rated themselves as "Intermediate" experienced in programming, who submitted solutions that fulfilled the requirements with 62.5%. This means that the "Novice" students had a distribution, with 44.5% critical and 55.5% failed submissions. Whereas the "Intermediate" students had 25% critical and 12.5% failed solution submissions.

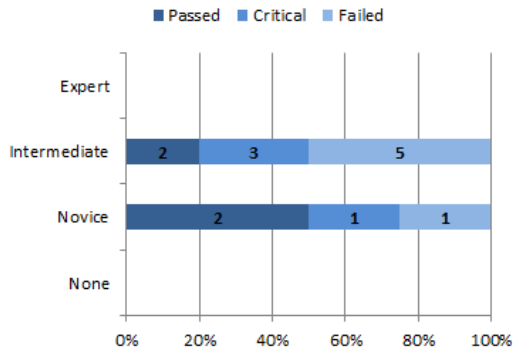

Figure 9: NSTF percentage distributions with relative task numbers

Figure 9 shows the distribution of the student's experience and states of the NSTF tasks. Here it is seen that both the "Intermediate" and the "Novice" students have submitted solutions that fulfilled the requirements, with a distribution of 20% and 50% respectively. Further the critical is distributed 30% and 25% where the failed is distributed 50% and 25% respectively for the "Intermediate" and the "Novice" students.

Figure 10 is a visualization of the total normalized data divided into the four different tasks. These data is calculated based on the weights: Passed=1, critical=0.5, failed=0.01.

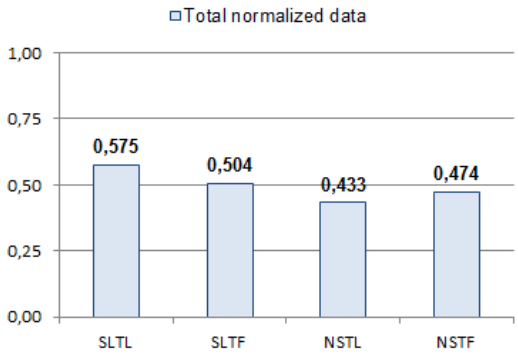

Figure 10: Total normalized data distributed in tasks

It is seen that the students who had the sliced tasks performed better than the students who had the non-sliced tasks.

Figure 11 shows "Novice" and "Intermediate" normalized data respectively, in accordance to the total normalized data.

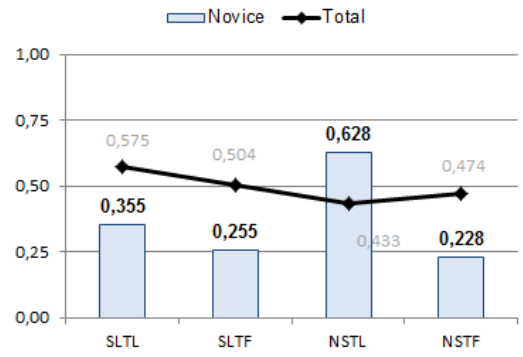

Figure 11: Novice normalized data distributed in tasks in accordance to total normalized data

In figure 11 it is illustrated that the "Novice" normalized data is lower than the total in all tasks but in the NSTL task. This data suggest that people who rate their programming experience as "Novice" is more predisposed to doing well when given the NSTL task, than when given one of the other tasks.

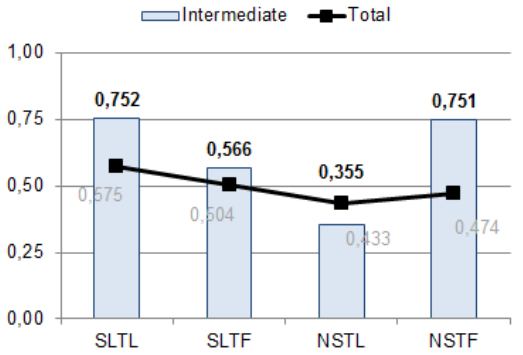

Figure 12: Intermediate normalized data ditsributed in tasks in accordance to total normalized data

Figure 12 shows that the "Intermediate" normalized data is higher than the total in all tasks but in the NSTL. Which suggest that people who rates their programming experience at "Intermediate" is more predisposed to do worse when given the NSTL tasks in comparison if given one of the other tasks.

5. DISCUSSION

In this section we compare TDD and CFTA according to our results. The result doesnâĂŹt give any clear answer to what is best, but the results indicate that for the students as novice developers it was easier to use the code first approach, which could be because they didnâĂŹt know how to solve the problem at first, but used trial and error to solve small parts of the assignment step by step to find a solution that seemed to cover the requirements. When the developer does not know how the code will be structured in the end it is difficult to write tests that will cover everything.

This has to be compared to the more experienced developers, who knew how they were going to solve the assignment from the requirements. Which allowed them to write the tests according to their initial solution and then use the tests to identify issues they might run into when implementing the logic later on.

But as all of the students are new to programming these are only assumptions, and we have to do more research to find a clear answer, as the results will most likely be different with a group of more experienced developers.

For the next iteration of this experiment it is paramount that the students are more clearly instructed in what the assignment is, to ensure that they are more focused on performing the experiment in accordance to description, as the validity of the results become questionable when the students differ from the described pattern of the experiment.

To ensure that there was no bias in the assignment of tasks the students received tasks randomly, this has the benefit that we as a group were unable to bias the results by handing out the assignments to the students in a way that would increase the probability of favourable result. The downside to this approach is that there was an unequal distribution of students across the different tasks, but with a sample group of this size it was not an issue.

6. THREATS TO VALIDITY

In this section we will go over the threats to validity using the guidelines by Wohlin et al.[4].

The threats to the Internal validity of this experiment are mostly related to the selection process. The experiment was conducted while the students were performing a test, which would indicate that they were highly motivated to perform well.

Further External threats to the validity were amongst others the *Interaction of setting and treatment*, as the test subjects were given small simple tasks to complete, which does not represent a real world setting the results from an experiment in the field might vary. *Interaction of history and treatment* is an important factor as the subjects had received lectures in all the topics tested in the experiment a few weeks up to the experiment. The participants were not representative of the target group (Software Developers) which also had an effect on the results, as they were unable to perform at the same level as the level as the target group.

One of the major Social threats was that the students talked about the tasks during the experiment, which was not intended, as this could hamper their individual performance. Another threat to the Social validity comes from the human discomfort from being evaluated, which can have influenced the results. The main *Hypothesis guessing* threat comes from the students recieving 5 weeks of TDD lectures just prior to the experiment, this might have impacted the subjects answers more in favor of TDD in the pre- and post questionnaires as they expected us to want them to be in favour of TDD.

The results might also have been affected by the studentâĂŹs not having the proper prerequisites ready before the

experiment, and therefore spent some of the experiment time on preparing their system. This might have had an impact on how much of the assignment they completed, but as stated in the experiment kit, they were not scores on how much of the assignment they handed in, it was focused on how well the handed in code worked as well as how well tested it was.

The principal Conclusion validity is affected by the *heterogeneity of the subjects* was rather large, as the test subjects was still only in the start of their university education, so their previously programming ability still played a major role in their programming proficiency which is also concerning the External validity. Also we do not claim that the results can be generalized for the entire software developer population, as these students were having difficulties solving the basic programming task they had to hand in.

7. CONCLUSION

In this study we performed an experiment to investigate if there was an increase in productivity when the using TDD instead of CFTA. The test subjects were not representative of the target group of software developers so these findings only reflect how TDD and CFTA compare when used by students.

But the experiment indicates that for students considering themselves as Novices making the tests first yielded worse results than Test Last. The students considering themselves as intermediate the Test first seemed to yield the best results. But these results has to be tested further to make any generalized conclusion as there is no clear trend in the results.

In the future we would like to perform a similar test on a group which has a closer relation to the actual target group. We would also investigate if it would be possible to use a metric to determine a subject's programming proficiency objectively instead of using the user's opinion. This will hopefully lead to a more general conclusion regarding the performance differences regarding TDD and CFTA.

8. REFERENCES

[1] D. Fucci, B. Turhan, M. Oivo, *On The Effects of Programming and Testing Skills on External Quality and Productivity in a Test-Driven Development Context*, Oulu, Finland.

[2] D. Fucci, B. Turhan, *A Replicated Experiment on the Effectiveness of Test-first Development*, Oulu, Finland, 2013.

[3] D. Fucci, B. Turhan, M. Oivo, *Impact of Process Conformance on the Effects of Test-driven Development*, Oulu, Finland, 2014.

[4] C. Wohlin, *Experimentation in Software Engineering: an Introduction*, volume 6. Springer, 2000.

9. ACKNOWLEDGMENTS

First of all, we wants to thank Dr. Marco Kuhrmann for his participation and help in this experiment. Especially for providing the necessary communication with the former of the experiment kit, and the provided tools and data.

We also wants to thank Davide Fucci, Burak Turhan, Markku Oivo from University of Oulu for providing the experiment kit used in this experiment.

Thanks to all students of 3. semester Software Engineering 2016 of University of Southern Denmark for participating in this project.