Working title

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

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

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

ABSTRACT

Background: Test Driven Development (TDD) is considered a good approach concerning productivity compared to the more common Code First Test After (CFTA) approach, and it works by implementing the tests first, and then by only writing enough logic to pass the test. *Goal:* We aim to investigate if there is a performance gain in using Test-Driven Development (TDD) over Code First Test After (CFTA). *Method:* We 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 with either step by step instructions (Sliced) or free text (Non-sliced) requirement descriptions. Further the subjects had to answer one questionnaire before and one after the coding task. *Result:* The experiment showed that with this test group the approach was less important than how detailed the task was described.

Conclusion: 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.

1. EXPERIMENT EXPLANATION

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

Before and after the experiment the students were handed a questionnaire; the first with questions regarding their perception of their own skill level, and the latter with questions about how well they thought they had performed during the experiment.

The experiment was split into 4 different tasks:

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

These tasks were randomly distributed among the students with no knowledge to the group of which students got which specific task.

At the beginning of the experiment the students received a code skeleton to work from, to ensure that the assignments would have a similar structure, and therefore it would be easier to compare the results. The code completeness and correctness was then 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. The code and questionnaires were then analysed to see if there was a correlation between testing before or testing after writing the code, and the task description's granularity.

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.

2. DESCRIPTIVE STATISTICS

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 for 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 attempts, where the solution had no chance to fulfill the requirements.
- 16 - critical state attempts, where the probability for the submission is high, but the fulfillment of the requirements is questionable.
- 23 - passed attempts, where the solutions were fulfilling the requirements.

DOI: 10.475/123_4

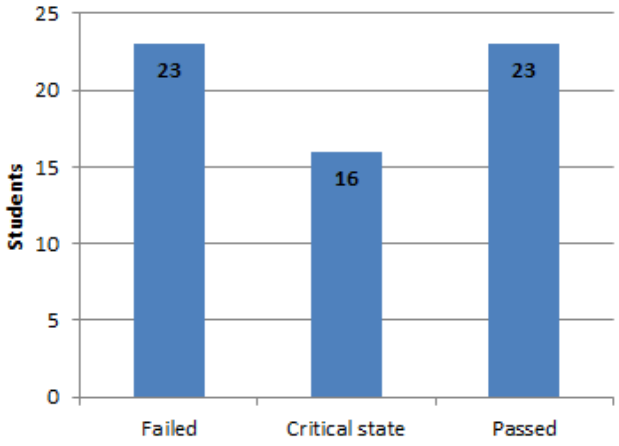

Figure 1: Distribution of student's performance

The diagram above 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 follows:

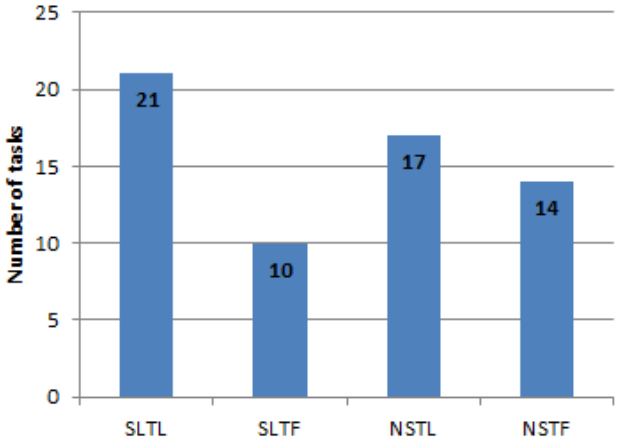

Figure 2: Task distributions

38 students were to do the experiment with the test-last approach, where 24 students had to do the test-first approach. Further it is an exceptionally case that both the sliced and non-sliced tasks were equally distributed with 31 of each.

After the assessment of the student submitted code, it is possible to look at the answers of the questionnaires. Shown in the following diagram is 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.

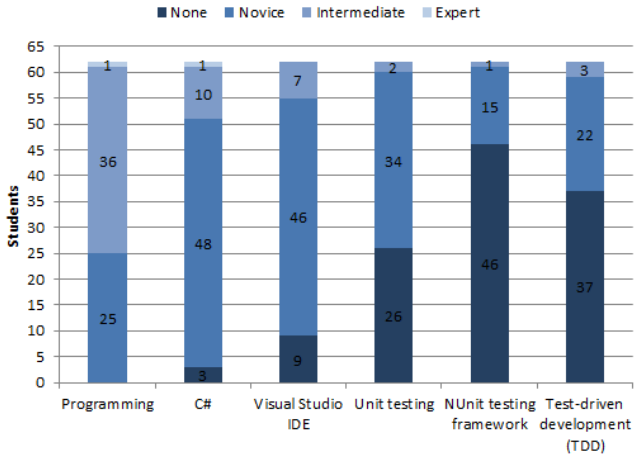

Figure 3: Student's own experience rating

It is important to see how the subject's self perception of their experience in TTD also seems to be based on which task the subject were handed.

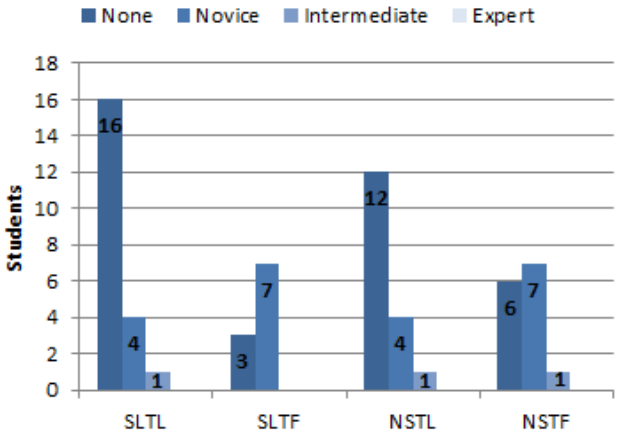

Figure 4: TDD experience in tasks

When looking at the above diagram, there are more people with none TDD-experience in the test-last approach, which also is an exceptionally 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 among the students, the following diagram shows the code states divided by the tasks.

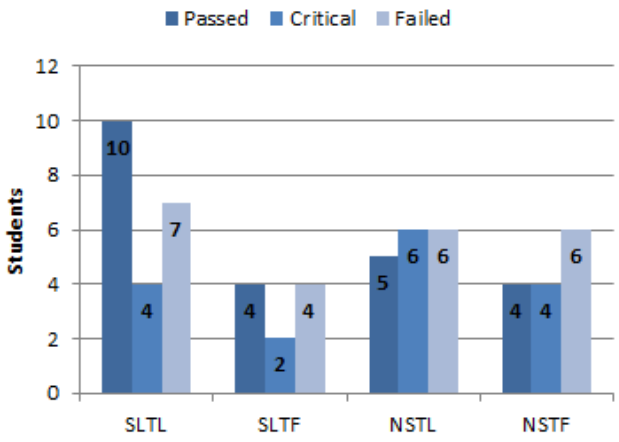

Figure 5: Student's task performance

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

3. FINDINGS

The following diagrams 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 we could have excluded this in the diagrams below. These are however included in the diagrams as it was an option in the pre-questionnaire, and therefore if excluded, would give a skewed view of the students programming experience levels. This does also apply for the "Expert" ratings which will be described below.

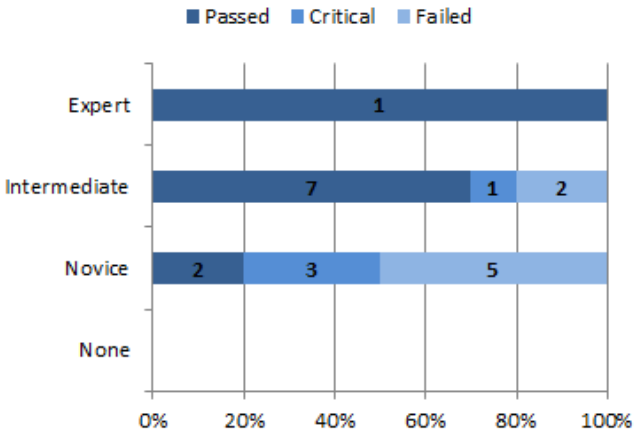

Figure 6: SLTL percentage distributions with relative task numbers

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.

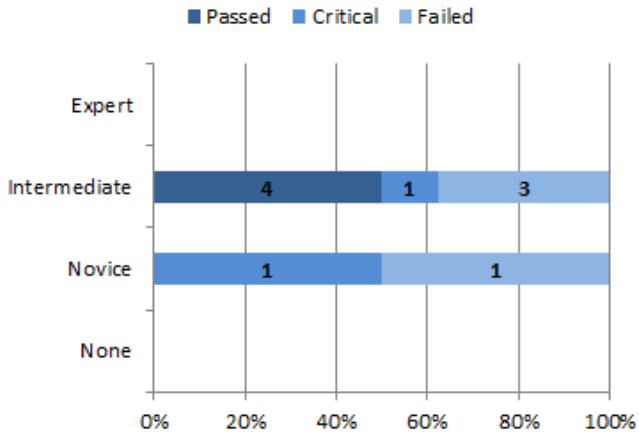

Figure 7: SLTF percentage distributions with relative task numbers

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 is 37.5% and 50% for respectively the "Intermediate" and the "Novice" students.

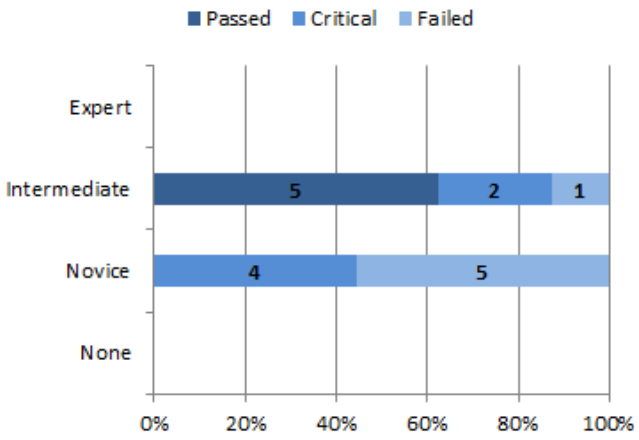

Figure 8: NSTL percentage distributions with relative task numbers

Above is 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.

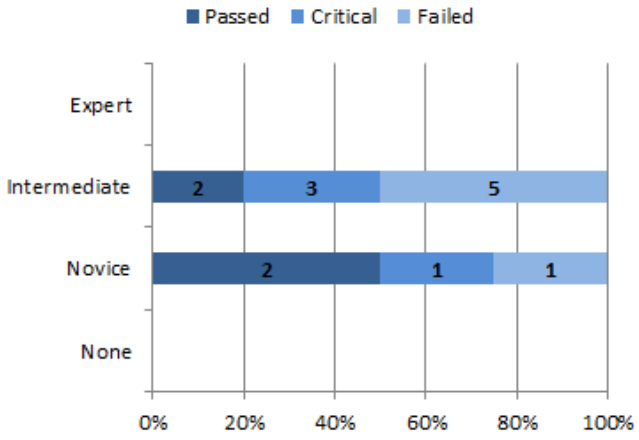

Figure 9: NSTF percentage distributions with relative task numbers

Above is 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.

The following diagram 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.

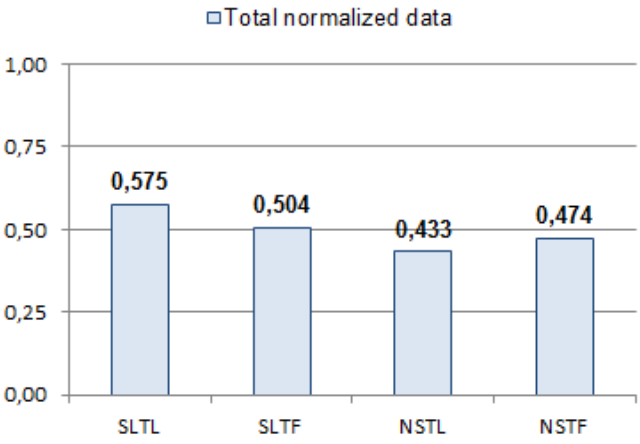

Figure 10: Total normalized data distributed in tasks

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

The following diagrams shows "Novice" and "Intermediate" normalized data respectively, in accordance to the total normalized data.

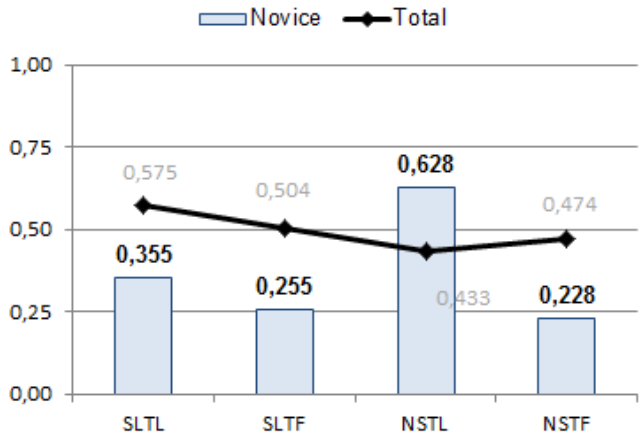

Figure 11: Novice normalized data distributed in tasks in accordance to total normalized data

The above diagram shows 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.

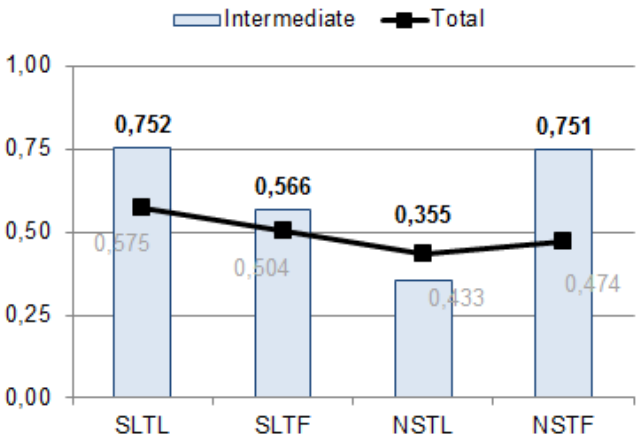

Figure 12: Intermediate normalized data ditsributed in tasks in accordance to total normalized data

The above diagram 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.

4. THREATS TO VALIDITY

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 were the subjects inability to work alone, as some talked together during the experiment, which was not intended. The group did not consider *Hypothesis guessing* a threat for the coding part of the experiment, but had to reevaluate it after the end of the experiment, as the subjects did not have more than 5 weeks of experience with TDD from received lectures. This might have impacted the subjects answers more or less in favor of TDD in the pre and post questionnaires. Another threat to the Social validity comes from the human discomfort from being evaluated, which can have influenced the results.

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. The conclusion validity is generally strong because of the high *Statistical Power* in having a large group of participants, but we do not claim that the results can be generalized for the entire software developer population.

5. DISCUSSION

When the group have conducted the experiment, it was clear that there were some aspects that could have been improved, or done differently. First of all the group had different roles during the experiment, as they both acted as supporters and observers for the experiment. The supporting role was expressed during the experiment, when the students had any questions regarding the experiment in general and when more specific problems appeared, when eg. their IDE did not work properly. Whereas the observation-role were at any other point, to keep track of how the students performed the experiment. These roles could have been divided so some members of the group only acted like supporters and others as observators, which could have giving the group a different point of view as well as both having the subjective and objective view on the experiment process.

The subjective view gave the group the knowledge that it seemed like the students did not actually read the experiment task properly, where the objective view gave the group knowledge that the students did talk together even though they specifically had been told not to.

The coding submissions were a direct result of the students programming level in general and their TTD experience was not as high as the group expected, based on their answers in the pre- and post questionnaires. Further the different tasks gives an idea that the sliced tasks were easier for the students to do than the non-sliced tasks, as the passed-level in these are lower than the critical and failure-levels.

6. CONCLUSIONS

7. ACKNOWLEDGMENTS

Thanks to Marco Kuhrmann
3'th semester bachelor Software Engineering students of SDU