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. RESULTS

In this experiment, 65 answers were submitted, but two participants submitted two versions, of the same questionnaire and one participant did not fulfill one of the questionnaire. A total number of 62 valid data points were collected.

As mentioned earlier, the participants were handed in two questionnaires, one before the task, and one after the task. The participants were asked to rate their programming abilities.

Their programming abilities were divided into 4 categories; none, novice, intermediate and expert. All abilities are measured per the number of years the participants have received lessons in programming. It is expected that each student has at least 1,5 years of programming experience, which is the measure used in the experiment.

- None - those with no programming abilities
- Novice - those with some, but small programming abilities (below average)
- Intermediate - those with the expected abilities per the level of experience
- Expert - those with good programming abilities (above average)

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Of the 62 valid data points, 11 overestimated their programming abilities, as the participants failed, but rated their abilities as intermediate or higher. Seven participants were in a critical state with an intermediate or higher rated programming abilities. This must also be considered as over estimation of abilities. 12 participants were failed and rated novice, whereas nine were in a critical state. 19 passed and were rated intermediate or higher. This must as those who rated their abilities, to match the outcome. Four was rated novice but passed. This must be considered those who underestimated their programming abilities.

The result of this experiment, was 62 submission of code, made in C#, 124 answered questionnaires, which will be used to conclude and visualize, the difference between

In the pre-questionnaire, the participants were asked about their background knowledge.

Currently working in industry:
Of all 62 valid submissions, 6 participants were already working or have been working in the industry. Of these 6 participants, 4 passed, 1 was in a critical state, and 1 failed the test.

3. CONCLUSIONS

4. ACKNOWLEDGMENTS