Testing Strategy

Dennis Thinh Tan Nguyen, Jacob Mullit Mniche,
Thor Valentin Aarkjr Olesen, William Diedrichsen Marstrand

3. december 2015

1 Introduction

We have chosen to do test driven development, because it helps to spot clunky and faulty code, which need to be refactored/improved early during implementation. This means that the code has to be modular since it will otherwise be hard to test against. The tests also forces you to get a clear idea of the constraints of the implementation that you are building. Since you are often running your tests, you also strengthen your belief in the system.

The TDD has been chosen even though some cons also exist: The maintenance of the test suite adds additional work to the implementation, but then again gives a higher belief in the functionality of the system and helps locating bugs early on, which seemed to be worth the extra work.

2 Unit Testing

We are mainly using unit testing to make it easier to detect and corrects faults, and to allow parallelism in the testing activities which is essential in our work, as every group member works on individual parts of the program simultaneously.

By testing all classes as they are developed, we make sure that all units are tested. The unit tests are automated which makes it easy to do regression testing, and in that way always have an idea of the robustness and quality of the program. We use black box testing to conduct tests of units before they are actually implemented. The reason for this is, that we know the functionality which the unit should support and as such we can create the tests based on this knowledge and check that our code does what is expected of it, by running the tests after the implementation. More specifically we use **equivalence testing** from black box testing to reduce the number of test cases. This has a huge impact in the cases where inputs and outputs are tested. In the equivalence testing we have give boundary testing high priority since it is often edge cases which make systems fail.

After a feature is fully implemented it is then marked as ready for white box testing. The white box testing should be conducted by a team member who did not work on the implementation. This approach has been chosen to let other team members get knowledge of the code which they themselves did not work with. Also the fact that the person did not work with the code gives a better chance that they might find faults during testing by trying things that the implementer('s) did not think about.

The goal for the white box testing is not to get path coverage, since this would be to comprehensive and not practically possible because of the many possible paths. Instead the goal is set to 90-95% branch coverage. The reason why this percentage is chosen and not 100% is that some cases will always be hard to cover, and thus this would require to many resources compared to the payoff from the tests.
The main goal is to test all major parts and all essential functionality of the program and a percentage of 90-95% would very likely achieve this.