COMP47480 Practical 3: JUnit Tutorial

Sukrat Kashyap (14200092)

27 February 2018

1 Work Done

In this practical, we had to use the TDD (Test driven development) approach to develop a piece of software. The first part required us to create a class which returns true for a temperature less than or equal to 0.0 and false otherwise. Since we had to use TDD approach we first created a class to test that the class function returns true. The test failed because we had no implementation the first time around so we implemented just enough code to pass the test on the second try. Then we wrote another test to see if the code returns false if the temperature is greater than zero. Since the test failed again, we wrote more production code to pass the test which completed our first part. The second part involved creating another software piece in which we pass the length of sides of the triangle and then check the type of the triangle with an isValid function. In the artefact pdf within this folder, one can see the step by step iteration of the TDD process that was employed. This basically involves writing a test which tests one part of the requirement and then writing production code to pass the test that previously failed. Code coverage was run to check how much of our test covered the production code. Since it left out a couple of branches in the "if else" condition, we created more tests to cover those conditions.

2 Reflections

Testing is a very important part of the development process. Many ways have been proposed to write and maintain the test. There are even many different types of testing for e.g. Black box testing, white box testing, integration testing etc. All of the types try to achieve the following objectives:

- To give the programmer the confidence in his code and hence removing the scary part when adding or updating a system.
- To keep in check of the bugs that have been found earlier.
- It does take more time in developing than only writing production code but it saves a lot of unnecessary need of debugging and finding failures later on.

- To ensure many properties of software like correctness, reliability, performance etc.

In this practical, we tried out one of the methodologies of testing known as TDD (Test driven development). Unlike traditional development, where we write code then write some test against it. In TDD, we write test according to the specification and then we write the code just to pass those test. It is followed by refactoring to clean up the code. In the practical, we followed this pattern to iteratively develop the Weather class in Part 1 and then we built the Triangle class according to the specification in Part 2. Each test case when written followed or tested a part of the specification. This part of the system was built in order to pass those test. No code more than sufficient to pass the failing test was written. Then in the next iteration, another test case was coded up which tested a different use case of the specification, followed by writing production code which passes the currently failing test without failing the previous ones. One can see the iteration in the artefact.pdf within this folder for the practical. Code coverage analysis was run later to check the level of code coverage by my test cases. It was a little below 100% as we left some branches in the 'if' condition for scalene and isosceles triangle due to the short-circuiting and combinatorial nature of boolean values. It seemed like maybe the TDD approach was not done properly. But in fact, the TDD was used correctly. It was just nature of the program, that required testing the same function by passing parameters in different order to pass through the short-circuiting and combinatorial nature of boolean expression. This was solved with the writing some additional tests.

To understand better about testing and assessing the test case (i.e each test case contribution to the coverage and checking a functionality). We tried to find a redundant test case by removing each case one at a time and checking the code coverage. The code coverage in my case did change whenever any test case was removed. Proving the contribution of the test case. We also tried to validate the test case by changing the method slightly from the specs which resulted in failing test assuring the contribution of the test case.

Overall, TDD seemed a bit tedious approach. But it proved to have a number of qualities. Because of TDD, we did not write a single line of unnecessary code which we usually do when designing and writing the code first. It also did not let us wander away from the specification into writing code thinking too much about the future. Since we were writing test's first it helped us to write code such that it was easily testable. The only downside that I felt was that the languages that we currently are using like JAVA do not help us with our TDD development because of its strict type in nature. Since we couldn't even compile our first test. Dynamic programming seems to support this TDD approach lot better. But at the same time, it makes the program too error-prone. Hence, TDD is an amazing approach but I feel we need a language that supports this methodology.