Project 2

CS 320 SNHU

Robert Sanford

The testing approach I used in this project was to directly align the tests with the specific requirements of the project, so that I could verify that each requirement had been met. For example, in Task.java there is a requirement that the contact ID could be no more than 10 characters. So, to test the code and make sure that it was met I used these lines of code in the TaskTest JUnit:

idTooLong = "111111111222222222333333333";

@Test

void getTaskIdTest() {

Task task = new Task(id);

Assertions.assertEquals(id, task.getTaskId());

Because the ID is too long in the test, it was possible to see if the code would function as it is supposed to and return an error message. I tried to ensure that each requirement of the project, such as input lengths and whether an input was unique, was not only tested, but also passed. Another example is in the Appointment Service, with the requirement that the date could not be in the past. Using the Date java utility, I implemented a test to ensure that the date entered was not before today’s date. Overall, the test coverage was good and tested most of the functions in the code as the coverage percentage was 89%. Having good test coverage allowed me to know that the tests were effective in testing the requirements of the project.

The JUnit tests and going back over the code often were measures that I took to make sure that the code was technically sound and functional.

The testing techniques I used in this project would fall under white-box and black-box techniques because the tests were both structure based, and specification based. The black box techniques included in this project were testing for valid inputs, conditions and actions, and events that generated outputs. The white box techniques used were to analyze components, path coverage, branch coverage, as well as the if then statements.

For the most part, I did not use experience-based techniques such as error guessing and exploratory testing. Error guessing, which relies on the experience and skills of the tester, was not used as I only used the JUnit tests to test the code. Exploratory testing is used, mostly, when time is a factor by utilizing “concurrent test design, test execution, test logging and learning within time boxes and is structured around the test objectives” (Hambling et al., 2015). In a sense time was a factor, but not to the extent that exploratory testing was used.

The mindset I adopted while working on this project was that of learning and experimenting, yet methodical. I am very new to software testing and JUnit tests, so I wanted to be sure I went through and addressed every requirement while also learning as much as possible. Caution was important in this project, and I utilized it by only testing and coding what the requirements were and not “overdoing” the tests or code. I tried my best to appreciate and understand the complexity and interrelationships in the code because of how it directly affects the overall quality of the final program. Regarding bias, I tried to limit it by testing all the requirements multiple times and not taking for granted the complexity of the project. At this point for myself, bias is not much of an issue as I am aware that I have much, much more to learn. In the future, as I become more proficient in software development, being aware of bias towards my work will be extremely important as it may be easy to get comfortable and over-confident, which is when errors would be more likely. If a software developer is responsible for testing their own code, it would be important to adopt the mindset of a tester and, in a sense, try to forget that they were the one who wrote the code.

Being disciplined in the commitment to quality as a software developer is extremely critical. In this course, we learned just how bad and significant the consequences can be for what may seem like even a small error in the code. Often, there are enormous amounts of money at stake as well as sometimes the health and well-being of people and society. Therefore, it is especially important to not cut corners when testing software and to be as thorough as possible because the ramifications of bad or buggy software can be huge.