Gunnar Dulle

CS-320

Project Two Code Summary and Reflections

**Summary**

When writing my tests for the three features in the app I based most of my tests on the software requirements. I checked to ensure that all the given variables were correctly filled by testing to ensure that in all cases a variable would accept being filled with a string the appropriate length and I checked that these variables would not accept a string that was too long, it would also not allow itself to be left empty either. An example of these tests can be found in the Appointment Service Test in lines 49-54 I test that the appointment description is fillable, in lines 64-67 I made sure that the description would not accept a null variable, and in lines 71-74 I tested to ensure that too long of a description would also not be accepted. I also made sure that the phone number variable in the contact class would not accept a variable that was too short in lines 81-91 of the Contact Service Test and ensured that the date variable in the Appointment class would not accept a date prior to the current date in lines 77-81 of the Appointment Service Class. Establishing this I also tested for each feature that the variables that needed to be alterable later would all change as needed which can be seen in lines 96-101 of the Task Service Test where I tested that the name of the task would be updateable. The overall quality of these tests was very good which was seen from the 100% test coverage I had with all my JUnit tests. This let me know that the tests that I had written were all effective in checking the code I had written and that the code was working as intended as well.

The JUnit tests helped me check that my code was technically sound as well, an example of this is in the Appointment Service Test in lines 104-113 where I wrote a test to make sure that the method of coming up with unique IDs for each appointment was sound. I generated three appointments and then compared the IDs to ensure that they were all unique, I used a similar technique in the Task Service Test as well to establish that what I perceived as the most technical aspect of the code I had written was working correctly.

My code became more efficient as I worked on it and refined my methods of testing. This was really seen going from writing the code for the contact feature to writing the task feature code. Going between these two features I began separating my tests out so that they were more readable and identifiable as to what was being tested. So, while the number of lines of code I wrote increased it became easier to see what aspects of the tests were failing and I could trace it to what part of the code needed to be revised much easier. An example of this change can be seen on lines 59-67 of the Contact Service Test and then lines 34-39, 51-54, 64-67, 96-101, and 112-116 of the Task Service Test. The section of the contact service test ran through all of the required tests for the first name variable of the contact feature, so it tested that the first name could be updated and was not too long and was filled. In the Task Service Test, I first tested that the task service name variable worked in lines 34-39, then that a null variable would be rejected on lines 51-54, that the name string was not too long in lines 64-67, that the name could be changed in lines 96-101 and finally that the name cannot be changed if the task is not found on lines 112-116. This separation of the tests withing my JUnit tests helped me to ensure that the code itself was running correctly and efficiently as the tests became much more efficient to read.

**Reflection**

When I wrote my tests, I used a few different techniques in order to ensure that my code was the best it could be. My techniques were predominantly specification-based but I also used some structure-based techniques as well. Specification-based techniques are those that are based on the given requirements for a project. These are the techniques that I relied on the most to make sure that the code that I had written was sufficient to meet the project requirements. Examples of my usage of these techniques would be my usage of the equivalence partitioning technique which assumes that inputs can be grouped with similar inputs so only one of these would need to be tested, I used this to test my address variable in the Contact Service Test. Since the test address I used was of an appropriate length I could assume that others would as well. I also used boundary value analysis which tests input values that border on the edge of the acceptable range of inputs to ensure that these edge cases fall into the correct group, I used this technique in my Task Test where I made sure that a task ID that was 11 characters long would not be accepted. I also used some structure-based techniques as well, these techniques are used to check that structures of the program are functioning correctly and not producing errors. I used this in both the Task Service Test and the Appointment Service Test when I tested to ensure that the IDs being generated were unique.

In writing my tests I did not use any experience-based techniques, these techniques are used by experienced coders when they don’t have the time to write specification-based or structure-based techniques. These techniques rely on what these coders have experienced in past projects to be able to make accurate assumptions about what tests a project will need. There are also other specification-based techniques that I didn’t use. I didn’t use decision table testing which uses tables to check that conditions based on different inputs and is helpful when combinations of inputs produce different actions, since this program does not have any condition changes this technique wasn’t necessary. I didn’t use state transition testing which checks when outputs are triggered by changes to the input’s state, the input’s state didn’t change in this program so this technique wasn’t necessary either. Use case testing makes sure that different actors in the system have correct functionality, this program doesn’t have multiple actors, so this technique wasn’t necessary either. This program also doesn’t have branching logic paths, so the structure-based technique of program flow charts wasn’t required either.

Specification-based techniques are integral to ensuring that a programs requirements are being met. These techniques take the given requirements and make tests based on them to check that the code reflects these requirements. Structure-based techniques check that the logic of the code itself doesn’t create errors in the program. These test cases come from the code itself to check that the code is working correctly, these techniques are also important in testing to make sure that a program will integrate into the system correctly as well. Experience-based techniques are techniques that an experience programmer can write based on experiences with similar programs. These are helpful when there is not enough time to write tests based on the given requirements or to parse through the code to write tests for the structure, this can speed up the testing process.

Working on this project I took a cautious approach working on this project, I spent time going over the requirements and made sure that I had code to meet the requirements for each feature. I initially was a little over cautious and I had written constructors for the features if they were missing input, but I later was guided to realizing that this was unnecessary and removed them. It was important to respect the code and its interconnected nature which led me to make sure that I had as many different variable names as I needed to make sure that things were not getting crossed in the code.

Limiting bias was very important in trying to review my code knowing how big of a factor this plays in developing code. I tried to write my code and tests on different days to break them up from one another in my mind. I also made sure to base my tests on requirements as much as possible and not reference the code itself to write my tests, the exception being when I felt like I needed to test to make sure that my code was not going to cause issues for the program. As a software developer I would try to do similar things if I were responsible for testing my own code. It would be important to acknowledge the inherent bias in testing my own code and do my best to make sure that I was really testing the code and not try to facilitate the code itself and test its strengths and ignore it’s weaknesses.

When developing code, it is extremely important to be disciplined in a commitment to quality in the software I produce. This means testing the code I’m writing as thoroughly as possible, testing it as hard as I can means that I also need to write the best code possible to stand up to sufficient testing. This all makes sure that the best product possible is being produced for the user. My goal to prevent technical debt is to work on developing some strong testing techniques now that can be replicated in the future to help prevent necessary testing from falling at the wayside.