Drew Pepin

Southern New Hampshire University

CS-320

August 21, 2022

7-2: Project Two

**Summary**

**To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**

There were several requirements I had to implement into my code throughout the course. The first being for ContactService.java, requiring the ability to delete contacts by using the contact ID of the contact you want to delete. If there is a matching contact ID with the one entered by the user for deletion, the contact is removed from the contact list (ContactService.java lines 36-48). The Task class required that the name object be less than 20 characters and cannot be null. To meet this requirement, I added a test that confirmed that the name object was less than 20 characters, and ensured the name wasn’t empty (TaskTest.java lines 29-34). The Appointment class required that the appointment date must not be a past date and cannot be null. To ensure this, I added a test that checks to make sure the date meets the requirements (AppointmentServiceTest.java lines 12-20).

**Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?**

The tests implemented for each class were highly effective in ensuring that all the requirements were met. The coverage percentages for all the files in the program were exceptionally high (above 91%). Based on the success of the tests and the high coverage percentages, the tests were effective in meeting the requirements implemented.

**How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

To ensure that my code was technically sound, I used good coding practices to keep my code consistent and easy to read, as well as testing every method appropriately. An example of a technically sound piece of code that I wrote would be in my ContactServiceTest.java. I implemented a test that ensured contacts could be added, deleted, and updated successfully. The test returned expected results, and I’m happy with how the code turned out and how well organized I made this test (ContactServiceTest.java lines 9-46).

**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

To ensure efficient code, I looked back at all my written classes and methods to ensure everything was as organized and compact as possible and removing any unnecessary lines of code. Some examples of efficient code would be my TaskServiceTest.java. All the tests for the requirements and methods are clean and compact, using minimal lines of code (TastServiceTest.java lines 14-124).

**Reflection**

**What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**

The testing techniques I implemented in this project were the checklist method and the use case method. Checklist based testing was implemented to ensure that the requirements were met properly that were implemented. The use case testing was implemented to ensure that when a user of our application performs an action, the system responds in a way that accomplishes that action.

**What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**

There are several other testing techniques that could’ve been used for this project or future projects, including risk-based testing and consultive testing. I don’t think these types of testing should be implemented into this project, as there weren’t any specific risks involved, and I had no team or product owners to consult.

**Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.**

For this project, I implemented caution in my testing methods extensively. With the different requirements and relationships between the classes, it’s important to carefully go over your tests several times and ensure everything is done properly and meets industry standards. Appreciating and understanding the complexity and relationships between your code is important, as methods will run and test differently when they are called between multiple classes. An example used in my code that portrays this is when I add a new appointment. This method calls another method that implements a unique ID to the appointment. These methods portray interrelationships between classes and methods (AppointmentService.java lines 17-19).

**Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.**

For me, implementing only what the requirements stated was the best way for me to approach having no biases within my code. I don’t think it was too difficult to avoid biases in this project, as the requirements were clearly listed, and the project as a whole didn’t have much opportunity to add code that would have biases within it. Bias would be a huge concern when you are responsible for your own code. Implementing testing methods based on biases and not to meet the requirements only can be a concern when developing.

**Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.**

Being disciplined is crucial when developing and testing your own code, but even more important when developing code within a team and for someone else. You can’t afford to cut corners when writing tests, as missing a key piece to a test can ruin a project and ruin your company and teams’ reputation. To avoid technical debt, I plan to test frequently and thoroughly, as well as plan my code and testing ahead of time before approaching a project.