Fernando Lomeli

CS 320

Prof. Omar Toledo Lopez

Project 2: Summary and Reflections Report

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.

During my work creating the contact, contact service, task, task service, appointment, and appointment service classes I made sure to prioritize the requirements. It was also a priority to test all the client requirements when writing the JUnit tests to make sure the requirements worked correctly. For the Contact class I made sure to not allow certain fields to be null and to keep them under a certain limit of characters such as the ID, name, and address fields. I also made it so that the phone number could only be added if it was exactly ten characters. For the Task class, I did not allow the ID, name, and description fields to be null or extend past a certain number of characters. For the Appointment class I ensured the appointmentID and description fields were not null and did not extend past a certain number of characters. I also made it so that the date field would only be allowed if it was a future date. I also followed the Service requirements by ensuring we could add contacts, tasks, and appointments, delete contacts, tasks, and appointments, and update contact/task fields via the unique ID.

* + 1. 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 quality of the JUnit tests for the contact service, the task service, and the appointment service classes were all very good as they all had 100% test coverage. Typically, good test coverage is over 80% so these JUnit tests excelled. There were not any missed instructions.

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.

I made sure that the code was technically sound by having the tests cover all the branches from the if/or statements. For example, I tested the name would not work if it was null or longer than 20 characters and that setting the name worked within the parameters while setting the name that was longer than 20 characters or null did not work. I also made sure the code was technically sound by ensuring all the requirements were tested. I kept the code simple and modular as I practically reused certain functions in different classes. The code also had some comments to explain to the user what was going on within the functions/code.

* + 1. How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.

I made sure my code was efficient by keeping the code simple, organized, and portable by using different functions to meet all the requirements rather than having everything cluttered and messy. This can be shown by the functions testNameIsNull, testNameTooLong, testSetName, testSetNameNull, and testSetNameTooLong. I also used proper naming conventions so that anyone viewing the code could know what each field is and what is happening within each function.

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

The software testing techniques that I employed for this project are JUnit testing to test the functional requirements, regression testing, and boundary value analysis. JUnit testing was used to test units or certain classes such as the contact, task, and appointment classes. With JUnit testing we can test certain inputs and outputs within that class or unit. We can even measure how much of the class we are testing by using test coverage which displays a percentage of how much of the class was tested. JUnit tests also make use of annotations which tell JUnit what to do. Examples of annotations include @Test which is used to run a public void test case and @Before which is used to run a method before each test method in the class. Assertions are used to check that the JUnit tests are returning the correct results with a pass or fail status. Regression testing occurred as well because whenever mistakes or bugs in the code were found and fixed, I would test the entire program again to ensure it still ran correctly. Boundary value analysis was involved when testing to ensure that certain objects such as name, ID, and description fields were kept under a certain number of characters. For example, “if (contactID == null || contactID.length()>10) {

throw new IllegalArgumentException("Invalid contact ID");” this code shows that the contactID must be within the boundary of 10 characters or less.

* + 1. What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.

Some software techniques not used for this project include automated and exploratory testing. Automated testing is a technique that uses tools executed by the computer without human testers. Automated testing works great with large test cases. Exploratory testing is when the tester explores the program to find mistakes or bugs without a structured test plan.

* + 1. For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.

Unit testing is great for testing entire classes such as the contact, task, and appointment classes. Different inputs and outputs can be tested within the functions in the class. Regression testing is essential for different software development projects because whenever changes are made to fix or update the code, the entire program should be tested to ensure it still runs correctly. Boundary value testing is important in software development projects because you may need to set a value limit to passwords, usernames, phone numbers, or other fields to ensure passwords are strong enough and to ensure certain fields are correct.

* 1. Mindset
     1. 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 my mindset was set on testing every possible input to ensure the code runs properly when initializing the fields or updating the fields. I was careful to ensure I tested all the requirements. It is important to appreciate the complexity and interrelationships of the code I was testing because the complexity was not too difficult which made it easier to test and the code being related made it easier to work with the different modules.

* + 1. 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.

In a way I limited the bias of my own code by not imagining it as my code but instead thinking of it as the clients work to reach their requirements. I was mainly focused on ensuring that I reached all the client’s software requirements during testing. I could imagine it to be difficult to review your own work because whenever I proofread my own essays in the past I would not find nearly as many errors as when someone else proofread my essay. You must push past it and focus on testing all the client’s requirements.

* + 1. 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.

When it comes to writing or testing code it is important not to cut corners because the code should be technically sound and of high quality or there could be issues or bugs that arise. This can be very costly if the code makes it into the live environment with defects. I plan to avoid technical debt as a practitioner in the field by not cutting corners and ensuring the code is readable, maintainable, and modular. The test coverage should be as high as possible, and code should be within proper coding standards such as proper naming conventions, error handling methods, and clear comments.