Nathaniel Burt  
CS 320  
4/19/2024

**Summary and Reflections Report**

**Summary:**

For each feature within the project, I utilized a systematic unit testing approach, as seen by the TaskServiceTest class and the other classes. I began by delving into the requirements to identify the functionalities and potential edge cases. For instance, in testing the TaskService, I created test cases to cover basic operations such as adding, deleting, and updating tasks, as well as handling exceptions for invalid inputs. This initial analysis formed the foundation for designing the test suites, ensuring that critical functionalities, like task deletion, were validated.

An essential aspect of my unit testing approach was its alignment with the software requirements, as demonstrated in the AppointmentServiceTest class. I frequently referred back to the requirements documentation to ensure that the tests adequately addressed the expected behaviors and scenarios. For example, in testing the AppointmentService, I verified that appointments could be added, deleted, and retrieved correctly, in accordance with the documented specifications. This validation of each requirement through targeted test cases built confidence in the correctness and completeness of the implemented features.

Quality plays a big role in the creation of my JUnit tests and was an important aspect of the project. I used code coverage tools to gauge the percentage of code covered by the tests, striving for a high coverage percentage to maximize test effectiveness. Additionally, I maintained a continuous cycle of test review and refactoring to ensure clarity, readability, and maintainability of the test suites. By upholding testing standards, I was able to deliver high-quality test suites that provided comprehensive coverage while minimizing overhead to the best of my ability.

Writing JUnit tests for Project One was an interesting experience that underlined the importance of technical soundness and efficiency. To ensure technical soundness, I created test cases that thoroughly exercised each method's behavior, as demonstrated in the ContactTest class. For instance, when testing the ContactService, I validated that contact information could be updated correctly by designing test cases for each field, such as first name, last name, phone number, and address. Here is an example of one the tests for a function in that class from that class:  
public void testAddContact() {

Contact contact = new Contact("456", "Jane", "Smith", "9876543210", "456 Elm St");

contactService.addContact(contact);

assertEquals(2, contactService.getContacts().size());

By optimizing test setup and minimizing redundant test cases, I ensured the efficiency of the test suites.

**Reflection:**

In this project, I utilized various testing techniques, including boundary value analysis, equivalence partitioning, and mutation testing, as illustrated across the test classes. Boundary value analysis allowed me to uncover potential off-by-one errors, while equivalence partitioning reduced redundant test cases. Additionally, mutation testing helped enhance test suite effectiveness by introducing intentional faults. While stress testing and load testing were not utilized in this project, they remain valuable techniques for assessing system scalability and performance under realistic deployment conditions.

My mindset throughout the project was characterized by repetitious testing and making sure I followed through with each requirement asked of me for each of the classes. I approached testing with the goal to fulfill said requirements, ensuring comprehensive coverage of all critical functionalities. I can’t say there were any biased opinions while making the code. I could see issues if the person you disliked in your company wrote the code, but then it would fall on you as the individual who’s testing it to ensure that it works properly. Either way, it’s important to ensure quality testing regardless of any biases.

I recognize the importance of discipline and quality commitment in software development, as shown in my code, I refused to cut corners and aimed to minimize technical debt and uphold the highest standards of software quality that I could. Through the testing and code review processes, my goal was to deliver software solutions that are reliable, maintainable, and poised for long-term success. Cutting corners introduces defects and vulnerabilities in the software. This in turn helps avoid technical debt by reducing the cost needed to fix major issues that could arise from the defects. Ways to combat this could include refactoring regularly and documenting your code thoroughly and always embrace continuous improvement.