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.

Each requirement was coded to be its own method. This makes testing each requirement easy because I can just test the different lines of logic in the method, name the test after that method, and include what logic I am testing in the name. If I do this for each method, I end up with a testing approach that tests each software requirement individually. For example, one of the Task Service requirements was to be able to add tasks with a unique ID. I tested what happened when adding tasks to an empty list, succeeding at adding a task to a list with other tasks, and failing at adding a task to a list with other tasks.

* + 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?

Usually, 80% or higher coverage is accepted as effective testing. I made sure that each test class was at least 80% coverage on the class it is testing.

* 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.

To make sure that my code was technically sound, I tested most of the decisions the classes make with descriptively named tests. For example, to test making an Appointment object, I split up the parameters into different tests that were named after the parameter they were testing. This ensures that people looking at my code and tests for the first time understand what I am testing.

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

To make sure that my code was efficient, each test class used a BeforeEach method to reduce the amount of redundant code. For example, a test that makes sure tasks can be added to the list in TaskServiceTest only needs to use addTask(task1) instead of making a whole new object.

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

I employed functional testing and unit testing to make sure my code was working as expected. With functional testing, I made sure that the code met the functional requirements that were set out by the client. Unit testing just ensured that each piece of the system was tested before they were combined into one system.

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

I did not use system testing, white-box testing, non-functional testing, or regression testing. System testing just makes sure all the units combined work as intended. White-box testing ensures that the structure of the code is up to standards. Non-functional testing makes sure that the non-functional requirements are included in the code. Regression testing tests the code after a change is made.

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

Functional testing is used in every project as it makes sure that the functional requirements work as intended. Non-functional testing is used in projects where security and speed are important to the system. Unit testing is used when there are multiple parts to a system that can be tested individually. System testing is used in projects that have multiple pieces that are combined into one system. White-box testing is used when the structure of the code or the flow of data matters.

* 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.

When testing the bounds that I made with my if statements, I made sure to test all the edge cases. If the complexity of the code is not realized before testing, a tester may do insufficient testing or too much testing making their tests unreliable or hard to maintain.

* + 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.

I indiscriminately tested most of the lines of logic in my code. I can see a developer taking pride in their work or using laziness as a reason not to test or fix a certain part of their code.

* + 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.

Cutting corners in software engineering can lead to harder to maintain code, problems that cost money to fix, and trust issues by clients. This can be seen in the recent recall of 169,000 Hyundai and Kia electric vehicles because of a software error (*Korea Times News*). I plan to confront myself and others when I see these issues arise.

References

Hyundai Motor, Kia to recall nearly 170,000 EVs over charging software error. (2024, March 14). *Korea Times News* [Seoul, Korea], NA. <https://link.gale.com/apps/doc/A786618191/GIC?u=nhc_main&sid=ebsco&xid=7c0f52fe>