**CS320 Project 2**

**Summary and Reflections Report**

Dante Trisciuzzi

4.12.2023

CS320 - SNHU

Prof. Albanie Bolton (Ph.D.)

**Summary**

During my work throughout the course of this project, my main focus was on ensuring that the classes met the requirements by writing tests. This approach, called test-driven development, involves testing throughout the development process. To do this, I first created a list of test requirements by outlining class structures and tests without writing the actual code. Additionally, I used static testing by reviewing the code before and after running the tests to ensure its quality.

Most of the testing I performed for these milestones was dynamic testing, which involved checking the variables for valid values and ensuring they did not exceed certain bounds. This helped prevent errors and abuse of the core application mechanics and objects. I used JUnit to execute the unit tests for dynamic testing, this in turn created more work items that needed testing. This ended up being a highly iterative process with the only hard end-point being the code coverage goal. This process helped me identify and fix code quality issues in the foundational components of the project.

The requirements of this project meant that some types of testing were not necessary or feasible. For example, we couldn't effectively test the security of the application because there is no primary application interface or user-facing components. We could add more unit tests in an attempt to test application security, but it would not make sense at this stage because the application is incomplete and we would not know what to test for. Moreover, we didn't perform any performance or usability testing, or any black-box testing in general (Morgan & Thompson, 2019).

While writing the jUnit tests for this application, I worked hard to make sure my code was technically sound. One of the most important factors in this process was to ensure the logic of each test was correct, before relying on it to test other code. This process could be tedious, but I followed the best practices outlined in the official jUnit documentation, including test structure. A good example of this can be seen on lines 25-36 of the TaskTest.java file; where I use a doc comment, jUnit annotation, and clear simple logic to test a small part of the application. This is also an example of another principal I tried to adhere to; testing in small ‘bite-sized’ pieces. This approach makes it difficult to write poor quality tests, since the tests should always have concise outcomes.

There was little to do in terms of code optimization in this project, however I did use modular methods in my class design. Two examples of this can be seen on lines 59-77 in the Appointment.java file. This block of code contains two private methods to check if a string is valid, and if a date is valid. The string check method takes several parameters to make it adaptable for different bounds checks and also checks for null, while the date check simply ensures that a date is not in the past or null. These methods greatly reduced what would have been a large amount of redundancy, since more than 50% of class methods needed to use this functionality. This design choice also made for code that was clean and readable throughout the remainder of the class.

**Reflection**

Working on a project with so few components can lead to feeling a bit too liberal with decisions, something I have tried to avoid. Throughout the course of working on this project I tried my best to consider the relationships between the object and service classes when designing the tests. This led to more concise tests that did not re-test functionality. For example, when testing the TaskService methods, I was careful not to re-test code that was being passed to the underlying Task object (De Rancourt Christian Stein, 2023). This would have little to no effect other than redundancy at this scope, but as I mentioned, this is one of the pitfalls that is easy to get trapped in with such a small codebase. This same concept if applied to a large model could end up increasing the testing time and complexity exponentially, wasting time and resources, while introducing more mode for potential bugs.

Limiting bias when reviewing or assessing any of your own work is difficult. However, it is much easier in a context like this where you have specific, immutable requirements that need to be met. The most important step I took to try and mitigate my own bias was starting with test design first as an integral part of early development. This helped me to focus on writing code to meet requirements, rather than just trying to design objects and service structures arbitrarily.

Software engineering is more than just writing code, I think that oftentimes the engineering aspect of this career gets overlooked. By this I mean software engineers are responsible for designing something that will be reliable and safe, the same way a structural engineer needs to design a bridge to meet or exceed standards. At first glance many people might not consider this, but as we have recently discussed in class, software bugs can lead to devastating consequences in the real world. Consider medical devices, where software runs on them 24/7, even something as simple as a rounding error or truncation could lead to an incorrect diagnosis, or worse. This was demonstrated by the infamous Therac-25 incident, simple software bugs (like the integer overflow) can quite literally kill people (Ethics Unwrapped, 2023).

As an individual, I set high goals for myself, and I take great pride in the work I do, but this is only possible because I also take accountability for my actions. As a software engineer, it is my duty to keep my work at or above standards. This is not as simple as it sounds; since standards are always changing, I need to stay up-to-date on any technologies I am using. Likewise, I need to never be too proud to ask for help or a second opinion. Vigilance can only take one so far in this field, because there is so much to know, and technologies are constantly evolving. I think I’m on the right path already, currently I use several version control tools with integrated CI/CD and dependency checking. In the future I’d like to automate even more of the development process, but until then, I will keep working hard to ensure my code is safe and secure.

**Resources**

De Rancourt Christian Stein, S. B. S. B. J. L. M. M. M. P. J. (2023). *JUnit 5 User Guide*. Retrieved April 13, 2023, from https://junit.org/junit5/docs/current/user-guide/

Ethics Unwrapped. (2023, February 15). *Therac-25 - Ethics Unwrapped*. https://ethicsunwrapped.utexas.edu/case-study/therac-25

Morgan, P., & Thompson, G. (2019). *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide*. BCS, The Chartered Institute for IT.

snhu. (2023). *Tutorial Using JUnit*. snhu.edu. https://learn.snhu.edu/content/enforced/1272209-CS-320-T4205-OL-TRAD-UG.23EW4/course\_documents/CS%20320%20Module%20Four%20Tutorial%20Using%20JUnit.pdf