**Austin Henley**

**CS-320**

**8/8/2023**

**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.

For each of the different features, I approached testing and creation of the functional code by planning out the requirements ahead of time. Tests were designed to ensure each restraint on inputs was being tested. With the Contact feature, for example, the test ‘testCeateContactWithNullIdFails’ was aligned to the requirement that the contact ID must not be null.

I followed the same methodology for each of the other features as well. After planning and creating the primary functional code, I designed the tests to check each parameter. Inside of the task test, I tested a boundary within the requirements with ‘testCreateTaskWithLongIdFails()’. This test was aligned with the requirement that the task object shall have an ID String that cannot be longer than 10 characters.

This was also seen in the appointment feature, with the test of ‘testCreateAppointmentWithPastDateFails()’. This test ensured that the code was following the requirement that the appointment date field cannot be in the past.

* + 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 overall quality of the Junit tests is high, based off the average coverage percentage of 91.85% across the six of my tests. The high coverage percentage shows that a large percentage of the code was actually tested. Along with the percentage being high, I also tested both pass and fail cases, to ensure the testing covered multiple use cases.

* 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 used many techniques to make sure I was utilizing the tools that JUnit provides. One of the methods I used in my code to ensure it was technically sound was Assertions. I used AssertThrows and AssertEquals throughout my tests to make sure that my features would fail when I expected them to fail. This can be seen in my TaskServiceTest class, in which I used assertEquals to make sure the task was named successfully and then I used AssertThrows to make sure it produced an error when the task does not exist in the service.

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

I ensured my code was efficient by using a beforeEach SetUp method to create preliminary objects. The goal was to reduce the code within each test to make them clearer and more precise, since the objects are created for every test anyways. I used BeforeEach in all my service tests:

A computer screen with text and symbols

Description automatically generated with medium confidence

A computer code on a black background

Description automatically generated

A computer code with text

Description automatically generated

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

In my project, I used software testing techniques such as unit testing and boundary value analysis. I used unit testing extensively, and focused on individual methods and requirements, and checked expected values against their actual output. I also verified that there were boundaries on each of the parameters in the code. This can be seen with tests like ‘testCreateContactWithLongDescriptionFails()’ in which I checked the outcome of an input at the edge of what is allowed in that method.

I also implemented other techniques throughout my testing classes. One of which is equivalence partitioning. Equivalence partitioning is shown when inputs are divided into classes and types (Software testing techniques, 2023). The usage of tests like ‘testCreateAppointmentWithNullIdFails()’ and ‘testCreateAppointmentWithLongIdFails()’ could be considered as testing different equivalence classes for the ID field, with null values, valid values and incorrect length values. Another is decision testing, which is seen when testing whether code makes the correct decision when faced with certain inputs. This is shown in my tests where I check if an exception is thrown by certain types of inputs.

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

Several techniques were not implemented in my testing. One of which is integration testing. This technique focuses on testing the interaction between the systems. My tests focused more on individual units (unit testing), so I did not test the interaction between different components. Another technique I did not implement was performance testing. This technique focuses on how the system behaves under certain conditions. I did not evaluate how this software performs under any type of load, but it could have been implemented by testing how well it performed by adding large amounts of objects (contacts, tasks, or appointments).

Specification-based techniques were also not included in this project. I did not implement techniques like decision tables and state transition testing. Both types of techniques are used to plan out expected outcomes and behaviors of the system. Additionally, I did not implement any structure-based testing techniques such as flow charts or statement testing. Flow charts visualize the software’s flow, and while my code has a flow to it, I did not design any flowcharts for testing in this project. Statement testing was also not used, and I did not test each individual statement in my tests, although my coverage was high.

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

Techniques used for testing are extremely important towards building a good final product. For instance, unit testing is important for the foundation of the project, since testing each method ensures the application works as expected. It helps with catching bugs earlier on and being able to repair them without breaking something else. Boundary value analysis tests the systems capability to handle and process input values outside of boundaries. This type of testing is especially crucial for user experience, to prevent crashes from unexpected inputs.

Equivalence partition helps improve the testing process by creating classes for different inputs. This helps make sure there is ample coverage, but also makes the testing process more efficient by reducing the number of required test cases. Decision testing helps guarantee the software’s ability to make correct decisions based on inputs.

Integration testing is important when considering the overall functionality of components working together. While unit testing ensures the components work, they may not work together well, and that is where integration testing ensures the different parts fit together. Performance testing gauges the software’s ability to handle stress from heavy traffic. It is important to find bottlenecks early in development for any front-facing application.

Specification-based techniques such as decision tables and state transition testing act as roadmaps for expected outcomes of the software. They are important for ensuring software meets user expectations and requirements. Along with those techniques, structure-based techniques help ensure code is accurately and efficiently tested. Tools like flow charts give a visual representation of data flow, and statement testing ensures comprehensive code execution. The end goal is reliable and efficient code.

* 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.

Consistently having a cautious mindset is important for testing. Each piece of code must be treated equally. Understanding the complexity of how each method interacts and the difference changes can make goes a long way towards creating a functional product. This is especially important considering that changing a single line of code in a class like ‘Task’ could affect all the services using it. It is important to consider all the effects the code can have.

* + 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.

It is easy to assume code is error-free if you wrote it. I find that when I test code immediately after writing it, it is harder for me to locate any errors. I approached the review of the code by stepping away for a moment and coming back to it. I approached testing and reviewing as if the code was written by someone else. Detaching from the code helps to be more objective in evaluating it. Testing your own code can especially be dangerous in terms of edge cases or scenarios that the developer did not originally think about.

* + 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.

Discipline is such an important part of creating quality software. Cutting corners does not equal long-term gain, and typically spending the time to focus on thorough tests pays off two-fold. I could imagine that skipping a “small” test could move you faster towards the finish-line but could also lead to an expensive and detrimental bug in production. I plan to avoid technical debt by always creating enough time to focus on testing. Testing can be one of the most important parts of the software development process. By testing thoroughly and continuously refactoring code, I can ensure the best effort is being put in for great software.

References

neeru360. (2023, February 6). *Software testing techniques*. GeeksforGeeks. https://www.geeksforgeeks.org/software-testing-techniques/