**Emily Bolyard**

**CS320 Project Two**

**12/12/24**

**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.

I made sure to try and look back at the requirements of the assignments that I had missed in the announcements to verify that I had all of the requirements correct. By going back over requirements it can help to make sure that your project is correctly aligned with the software requirements the client may be looking for. For example in the TaskService class I had missed in the announcement about not setting the currentIdNum to 0, this allowed me to go back and fix my mistake to better align with requirements.

* + 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?

Based on the coverage percentage using the coverage tool I was at 92%. This is a very high coverage percentage meaning that my tests were able to run how I wanted them to. This told me that the exceptions were thrown when they should be, and that the edge cases were being validated.

* 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.

We can look at the contact test class and see that I have tests in place for each value to check for null values as in the example, but that I also have tests for each requirement that was given.

A screen shot of a computer program

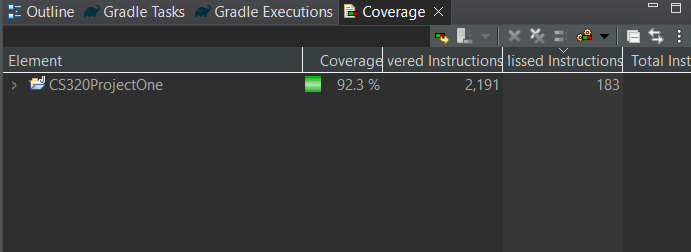
Description automatically generated

* + 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 ensuring that there were no failed tests during the testing phase after writing my code. I also made sure that the test coverage was above 90%, as close to 100% as possible. This allowed me to check the efficiency and correctness of my code.

A screenshot of a computer

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.

For the project I used some static testing and unit testing. Both types of tests are a form of white box testing since I was the one writing the code. This was the best testing method to use. Unit testing is where individual units of the software are tested in isolation from the rest of the system. This involves writing test cases for specific cases to test different input values. It is a method with fast execution due to testing small bits of code at a time, and good code coverage since the individual units are being tested not the software as a whole. For the unit testing I used JUnit testing. Static testing is the evaluation of the code by analyzing the code without running the program. This is something that should be done to code as you go; by analyzing small sections at a time you may be able to catch errors before you even have to run your program. This allows for early detection of errors and a focus on code quality.

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

One software testing technique I didn’t use would be security testing or any non-functional testing, since the code isn’t being turned into a functioning application this testing wouldn’t be necessary. Non-functional testing includes running security tests, reliability tests, performance tests, and usability tests. Another form of testing I did not use would be integration testing, which is tests run on the system as a whole by bringing all of the small parts together and testing all of them at once. Integration testing focuses more on the interactions between different components of the system that were developed separately and then combined.

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

Unit testing is important as it can help to cut down on errors early on and make the rest of the process run more smoothly. Unit testing can help with any missed logic and ensure that there are no null values or inappropriate string lengths. Static testing is helpful when doing small sections of code as well as it helps to catch any errors in mistyping and code quality. This can be used to help cut down on costs and time later on in the project. Non-functional testing is extremely important later on in the project whenever the application becomes a functioning product to ensure the security and functionality of the application before it is released. Finally integration testing would also be used on a functioning application to ensure that is working properly as a whole and not just when running in smaller sections. While the latter half of the testing implementations hasn’t been essential for the milestones so far, in a situation where you are developing a product for a client these are all extremely important tests to run to ensure maximum quality for the code you are developing.

* 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.

I adopted a more cautious mindset during this project since in my previous milestones I missed a requirement. This made me employ a bit more caution when assessing my code and validating that it followed the correct structure and requirements given.

* + 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.

Bias would be a concern if I were to assume that since I coded it that it would work before testing. Bias may also be a factor if I were to assume my code worked better the way that I did it instead of another way it may be coded. I am always cautious to assume that my code may not work so it is always extremely important to me to test my code often. For this project I often went through my code to ensure that I was meeting high-test coverage since I was worried, I may not be testing correctly.

* + 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.

Being disciplined when writing any code or tests is extremely important. Cutting corners when it comes to testing and coding could mean you make a grave mistake in the long run jeopardizing the entire project which could cost your company a lot of time and budget. To avoid technical debt it is best to write clear and concise code and tests without cutting any corners.