Chad Nadeau

Project 2

CS-320

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.***

At first, I had created my code based on previous knowledge and research, but as I received my feedback, I realized that I had not met the requirements properly. When it came time for the final project, I had gone through my code with a fine toothcomb to make sure that it followed the requirements and still made sense.

* + 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?***

Although I did not have perfect coverage, I still had sound coverage at around 70%. My coverage before making all of the changes I did was much lower, which I feel as why my tests are that much more sound.

* 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.***

Through out my tests, I had several tests such as the ones below to prove success and failures.

*@Test*

public void testDeleteTask\_Success() {

TaskService taskService = new TaskService(5);

Task task = new Task("T001", "Task 1", "Description 1");

taskService.addTask(task);

taskService.deleteTask("T001");

*assertNull*(taskService.getTaskById("T001"));

}

*@Test*

public void testDeleteTask\_Failure\_TaskNotFound() {

TaskService taskService = new TaskService(5);

*assertThrows*(IllegalArgumentException.class, () -> {

taskService.deleteTask("T001");

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

In order to make sure that my code was efficient, I made sure to test all aspects of items that needed to be updated.

*@Test*

public void testCreateTask\_Failure\_InvalidTaskId() {

*assertThrows*(IllegalArgumentException.class, () -> {

new Task(null, "Task", "Description");

});

*assertThrows*(IllegalArgumentException.class, () -> {

new Task("InvalidTaskId", "Task", "Description");

});

}

*@Test*

public void testCreateTask\_Failure\_InvalidName() {

*assertThrows*(IllegalArgumentException.class, () -> {

new Task("T001", null, "Description");

});

*assertThrows*(IllegalArgumentException.class, () -> {

new Task("T001", "InvalidTaskNameWithTooLongLength", "Description");

});

}

*@Test*

public void testCreateTask\_Failure\_InvalidDescription() {

*assertThrows*(IllegalArgumentException.class, () -> {

new Task("T001", "Task", null);

});

*assertThrows*(IllegalArgumentException.class, () -> {

new Task("T001", "Task", "InvalidTaskDescriptionWithTooLongLengthThatExceedsLimit");

});

}

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

Unit testing was involved which tested individual units or components of the software to ensure their correctness and focus on small, isolated parts of the code. Functional testing verified whether the software functions according to the specified requirements. Finally, with boundary testing, it involved testing the software's behavior at the boundaries of input values to ensure it handles edge cases correctly.

* + 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 mutation testing is that it introduces small changes in the code and verifies if the tests can detect these changes, which I didn’t see as a fit. As for performance testing, it evaluates how well the software performs under various conditions, including load, stress, and scalability testing, and with a small code base, I did not feel it was necessary. Since there is no user interface just yet, I did not include any usability testing.

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

For complex enterprise software applications, a combination of unit testing, functional testing, and performance testing is essential. Unit tests ensure individual modules work correctly, functional tests validate features meet business requirements, and performance tests confirm the software can handle high loads. Boundary testing is crucial for financial software to ensure accurate calculations, while mutation testing can be employed to verify the stability of financial algorithms. Functional testing ensures compliance with regulatory requirements.

* 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.***

Throughout this project, I tried to adopt the mindset of being in the real world, a real world where the software I was creating could be changing lives. Rushing through testing without proper caution could result in overlooking critical bugs that might have serious consequences in the production environment. Understanding the complexity and interrelationships of the code being tested is essential for effective software testing. Software systems are often intricate, with various components interacting in complex ways. These complexities need to be appreciated to design comprehensive test cases that cover different scenarios and interactions.

* + 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 in code reviews can arise from various factors, such as personal preferences, familiarity with the developer, or unconscious biases. I tried to avoid bias by taking a break from my code for an extended period and committing myself to something else. That way, when I came back to it, I had a new perspective when reviewing my code. Reviewing the feedback, I was provided with throughout the modules was a great way to also avoid bias.

* + 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 in my commitment to quality as a software engineering professional is of extreme importance for several reasons. It ensures the reliability, maintainability, and overall success of software projects. Cutting corners when writing or testing code can lead to long-term negative consequences, such as increased defects, reduced software performance, and the accumulation of technical debt. Never mind the fact that it could cause life-altering consequences to someone's life if you are not disciplined in your development.