**Project Two**

Michael D’Amico

Southern New Hampshire University

CS-320

Angelo Luo

April 21, 2024

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.  
           
         **In this project, I first created my task class with the 3 required variables. In the constructor there are three statements (one per variable) that throw in exception when they do not meet the requirements for that variable. These are the same requirements that are tested against in the taskTest.java file by doing “white box” testing. These unit tests are testing the method to ensure only valid objects are created from the “task” method. When creating the unit tests to verify requirements for “taskId”, I first sent in valid data and confirmed it was working in general. The second test was to confirm that it does in fact throw an exception when a null task ID is sent into the constructor. The third test was to confirm that if a task ID of greater than 10 was used to create a task, that it would throw an exception.**
      2. 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?  
           
         **When I ran all the tests, eclipse displays 100% coverage on both “task.java” and “taskService.java”. Beyond this, I have tested each requirement that it works and does not work as specified.**
   2. 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.  
           
         **Again, I designed unit tests to “break” the constructor by sending in known bad data and verifying the exception using the correct assertion. I also verified known good data to verify it was working.**

A screen shot of a computer

Description automatically generated**I did not have to test whether a taskID was updated because I did not create a setter for it in the class. In my taskServiceTest file, I do a test to store a taskID with the already used taskID and verifty that throws an exception.**

* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.  
         
       **I attempted to write the least number of tests to get full coverage and test edge cases. I used simple names for my tests to ensure that they would be quick and easy to understand. This will save time later.**

A computer code on a black background

Description automatically generated**In the picture below, I have also used a simple comment to show the variable was too large with 21 characters. This could help future testers quickly understand what is being tested.**

1. **Reflection**
   1. Testing Techniques
      1. What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.  
          **The type of software testing in these milestones was mainly white box unit testing. This is where a function is called, and parameters are sent in to verify expected outputs. The conditions assessed in my milestones were if they worked or didn’t work as per requirements and edge cases were tested to ensure thresholds in the requirements were not exceeded. The two files were visually inspected after creation to ensure there were no errors recognized by my IDE and to double check the requirements were met.**
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

**Integration testing-In these milestones we tested individual methods. Integration testing is where the system is tested to verify it works together.**

**Black Box testing-This is the type of testing that is done without knowledge of the code. This would include running the program and manually testing the system.**

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

**For integration testing, the system is being tested more in line with the way a user would be using the system. They would not be using the system as individual methods but the system. This is good for systems that are customer facing.**

**One of the practical uses of black box testing is that you could catch incorrect behaviors that would not be obvious through unit testing like an error message that does not make sense. A situation where black box testing is beneficial is UI testing. This is another layer of validation and another chance to correct problems.**

* 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.  
          
        **My mindset while I worked on this project was to always be looking for errors and to test every possible branch of the code/edge cases. I employed caution by making sure each requirement was met. It was important to appreciate the complexity and interrelationships of the code I was testing because I needed to understand that the code allowed for many different behaviors based upon what the user did. For example, if you create a task, there are multiple things that can go wrong. The name can be null or too long, the id can be null or too long, the task can be valid, and many more cases.**
     2. 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 tried to limit bias in my review of the code by testing based upon the requirements, not by the behavior of the code or my own interpretation. Thinking that I would have coded perfectly and without any errors is a flawed way of thinking. I can imagine that bias would be a concern if I was responsible for testing my own code. I may not think of edge cases that someone else would. I may also make assumptions that another person would not make.**
     3. 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.  
          
        **It is important to be disciplined because you need to carefully consider and understand the requirements. It is important not to cut corners when writing or testing code because the tests will continue to be used by other developers in the future. Poorly written tests may lead to bugs that are hard to find. Also, testing is a form of in-code documentation. If they are not written properly, they may confuse other people. I plan to avoid technical debt in the field by testing my code as I write it and by following the documented requirements. I would also set up a testing plan with the people I am working with before I start coding.**