Zac McBride

CS 320

4/15/2023

Project Two

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.    **My approach to testing was very much based on the requirements. I tried to make sure that a few tests were done for each requirement. I at least made sure there was a test within the requirement and one on the outside. For example, when a string needed to be under a certain number of characters, I had one test to make sure that an acceptable string would be taken and another test that had an invalid string with one too many characters. It can be assumed that if the string with one too many characters throw an exception, then other strings that are more characters would also throw an exception. A third test was done to ensure that null entries were not accepted as well. Other requirements needed more than these three tests. In the case of a phone number, the string had to be exactly ten characters and needed to be all numbers, so a test was done to ensure a string that was too short would throw an exception and another test made sure that only numbers were accepted. The “service” classes were also tested to ensure that an object of the respective type could be added, deleted, and be altered with the list created be the “service” class.**
      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?  **Each requirement used a new line of code to make sure that requirement was fulfilled. If a new line of code was written for the application, one or more tests had to be done on that line to ensure a high coverage percentage. While making tests I made sure to use the coverage percentage to see if enough of the code was being tested. Assuming that each test was effective, this would ensure that the testing done to the application was done as thoroughly as possible. The overall coverage percentage I achieved was 87%. This was enough above the 80% threshold that I was satisfied with the testing done.**
   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. **To make sure that the requirements were well tested, I made sure to write a test for each requirement. I would systematically look at one requirement at a time and write the tests needed to test that requirement. As mentioned above multiple tests were created to test a valid input and then to test the borders of where invalid inputs would start. Testing that an exception would be thrown if the string was too long or null and in other cases checking other requirements. Testing requirements for the contact, appointment, and task classes in this way was pretty simple. Testing the “service” classes relating to these was a little more complex to make sure that the requirements were tested well. I needed to check that items could be added to the list and that if an item was added or changed that did not have a valid input then an exception would be thrown here as well.**
      2. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate. **Testing the edges of acceptable inputs would make sure that the desired outcome would be achieved. Assuming that if a string that was one character is not accepted by the application, then strings of longer characters would also be rejected is a way to make the testing is done efficiently. The testing that needed to be done for the “service” classes had some other techniques to be efficient in the code for testing. I used an @BeforeEach function to set up what was needed for each of the tests. This used significantly less lines of code.**
2. **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 most part the testing done for this project was black-box testing. Notably boundary value testing and equivalence partition techniques were the most useful in designing tests for these requirements. Boundary value testing was already discussed above where an invalid input was tested on the boundary of what would be acceptable. Equivalence partition techniques chooses values to test that are on partitions that are acceptable or not. For example, if a string needed to be within a certain character length. Values would be chosen just within the boundary, just without the boundary and some value in the middle. I would need to create more tests that would have more values to fully utilize this technique.**
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details. **For this application the above techniques were sufficient for testing the requirements. There are some other black-box techniques that could be used for testing other requirements. Use case testing tests the functionality for how the users will use the system. Decision table testing creates a table for different outcomes based on certain inputs and testing whether these outcomes occur. State transition testing tests for when the state of a program changes when expected. These techniques would be more useful for testing the menus of this program which have not yet been created.**
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations. **Boundary and equivalence partition testing are useful for testing inputs that need to have certain parameters. This should be considered in all software programs as it will prevent security breaches using overflow to break a program. Use case testing is applicable in most applications as well, though this is going to focus on the parts of a system that the user interacts with not so much how the objects within a system interact with each other. Decision table testing is needed when particular outcomes happen based on what is input. This can be as simple as making sure that a page opens when a particular icon is clicked and may be complex with multiple things that need to happen to have an outcome. For example, a online shop would need to have items in a cart purchased by the customer with an address, and get the correct payment to create a shipment. State transition testing will only be required when states of a system need to change. This would be common when testing machines or appliances. An example would be testing a stoplight to ensure that the right color is given at any given time based on the circumstance.**
   2. 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. **This was an interesting project to write tests for as I was both writing the application code and the tests. I see why it would be beneficial to have different people write the code and the tests for the code. I found myself wanting to write tests for the requirements I had already had code for. This would make missing requirements very easy. This is why I had to go back through the requirements one at a time and think about what tests I needed to write to ensure that they were fulfilled. I realized this from the feedback of the first milestone. I had missed checking for the phone number being all digits. Part of the cation necessary for writing good tests is to write them for the requirements specified and not for the code that is already there.**
      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. **Again it is a lot easier to test the code rather than the requirements when you are writing tests for your own code. Once I started looking at the requirements and thinking about how I would need to test them, it was easier to write the code necessary to fulfill those requirements. I found that I had to change or add to the code I already had to make sure it passed a test created for a requirement. Again, the example of the phone number and making sure that it was all digits.**
      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. **Learning about the cost defects can have in a program that has been released is eye opening. It can really cost a lot, not just financially but in other ways as well. The worst of which being the cost of lives. Even the cost of brand perception can be costly. This is why writing quality code is very important. While working on this project, I have realized just how difficult this can be at times. Cutting corners can be tempting especially when a deadline is approaching, or expectations are not quite clear. This is unacceptable in any project. Testing is a way to ensure that quality is achieved. It is tangible proof of the quality of the code. I think any work done on software is something that anyone should be proud of. It takes a different kind of thinking and is an impressive skill. This pride is what drives me to write the best code I possibly can. I want the work I do to be meaningful and part of that is making ethical decisions while doing said work.**