**Project Two Summary and Reflection Report**

**CS-320**

**Laura McAroy**

**October 13, 2022**

**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.
         * Per the software requirements, the Contact class needed to contain a first name field that could not be null or more than 10 characters, a last name field that could not be null or more than 10 characters, a phone number field that could not be null or anything other than 10 characters, and an address field that could not be null or greater than 30 characters. All of the input requirements were validated before being assigned to the field.
         * According to the requirements of the Task class, each task needed to contain a name that could not be null or greater than 20 characters, and a description that could not be null or contain more than 50 characters. These input requirements were also validated before the task was created.
         * Per the requirements of the Appointment class, each appointment needed to be set for a date that could not be in the past, and appointments needed to include a description that could not be null or greater than 50 characters. All of the input requirements were validated before an appointment could be created.
         * All three classes also needed a unique ID to be created for each instance that could not be greater than 10 characters and could not be updatable. A random UUID was used to ensure that unique IDs were assigned to each instance, and no functions were written to update the ID once it had been created.
      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?
         * Writing effective JUnit tests with enough coverage took some practice. For the final application, we can see that the tests were effective by looking at the coverage for each test. The Appointment Service test had the highest amount of coverage at 92 percent, and the Contact Service class had the lowest at 81.5 percent. While it would be desirable for these numbers to be higher, since they are all over 80 percent, we can determine that the tests are effective.
   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 ensure that my code was technically sound, I implemented the input verifications in the mutators, like setName, setDescription, setPhone, and setAddress. By doing this, I could call on those functions throughout the program and the input would be verified every time. This design also adopts the principle of code reuse and avoids code repetition. I also used JUnit assertions like AssertEquals, AssertThrows, and AssertTrue in my tests to ensure the code was doing what it should.
      2. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.
         * To ensure my code was efficient, I ran my code frequently to ensure it was working correctly. I utilized an array in each of the three service classes and used a while loop

while ( i < contactList.size() ) {

if ( id.equals(contactList.get(i).getId())

{return contactList.get(i)}}

to search through the Array for a matching ID. I also utilized the randomUUID function to ensure that each contact, task, and appointment had a unique ID and there was no chance of duplication.

In making sure my JUnit tests were efficient, I created a separate test for each requirement. For example, I created a test and contained a task name that was too long to be accepted to ensure that an exception would be thrown like it should. Another test was created with a null name to ensure that an exception would be thrown for that as well.

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 this project I implemented both black-box and white-box testing techniques. Equivalence partitioning is an example of a black-box testing technique that was used, for the description fields input values less than 50 characters were accepted, and values longer than 50 characters were not. Structure based techniques (white box), would include tests that determined the accuracy of the if then statements, and the accuracy of the while loops that searched the size of the array, which are essential in meeting the input requirements set by the company.
      2. 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 any experience-based techniques, like error guessing, for this project, as I do not have any experience to fall back on. System testing, testing done after integrating new aspects of the system to ensure they all work properly together was not required for the project. Stress testing, to ensure that the program will still operate correctly under heavy loads, was not performed for the assignment either.
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
         * The use of multiple testing techniques is necessary in ensuring that you are creating a quality project. The more testing that is done, the more you can be certain that the work you are deploying is doing everything that it should, and nothing that it should not. Static testing ensures there are no design flaws before the application development starts. Dynamic testing ensures that the code delivers the correct results and behaves correctly in many different instances. If we do not include JUnit tests, we are skipping a critical step in assuring the overall quality of our application.
   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.
         * The mindset that I adapted while working on this project was one of a tester as opposed to a developer. That is, assuming that there are errors present instead of assuming that the code will run correctly without testing. I employed caution by creating test cases for nearly every scenario listed in the requirements, null fields and fields with too many characters to be specific. It is important to appreciate the complexity and interrelationships of the code because that is what creates a quality program. Making sure that all of the components of a system work together seamlessly is essential in creating a working product. In my first assignment, I only created one test case for the class. Obviously, this is nowhere near what was needed in order to accurately evaluate the functionality of the project. As I progressed to the final project, my testing became more thorough and complex to assure that all aspects of the requirements were covered in each class and that my code was technically sound.
      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.
         * Bias is most definitely present when a developer tests their own code. As I mentioned above, when a developer writes their code, that assume that everything they wrote is correct, otherwise they wouldn’t have written it that way. A tester must go in with the mindset that there are errors to be found and set out to find as many as they can. This would be difficult to do if you don’t believe the code has errors in it to begin with. Personally, I don’t have much confidence in my coding skills, so it was not difficult for me to assume that the code had errors. I tested each portion of my final project multiple times before submitting it, and made several changes to correct the errors that were found each time.
      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.
         * Commitment to quality should be of the highest importance when it comes to writing or testing code. There are too many tragedies, Therac-25, Mariner 1, and Ariane 5, to name a few, where people have lost their lives (or billions of dollars) due to lack of testing and inefficient coding. As with most jobs, developers and testers have an ethical responsibility to ensure that what they are putting out into the world will be beneficial. In my opinion, using a agile methodology is the best way to avoid technical debt. Agile methods are more flexible in that they allow for frequent planning and changes to made within the SDLC, whereas with a traditional waterfall method, errors may not be found until the final stages of development and could cost a lot of time and money to go back and correct.