Justin Lombardi

CS – 320

April 15, 2023

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. **I did feel that my testing approach aligned with the software requirements well. I believe that due to some errors in my testing, this dropped the integrity of my code to below the 80% mark which allowed for gaps in my tests. While I do believe that my approach was aligned with the requirements, I believe that I may not have been hard enough on my tests, especially when they were causing errors.**
      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?  **I believed that my JUnit tests were effective based on the logic I built them with and the approach I took to build them. I do understand that some errors occurred as stated previously that dropped the coverage percentage. I believed that they were effective because the methods held the correct requirements, but I later realized that some of my steps were missing valuable pieces such as not verifying a specific ID in a list, making an ID updatable, and verifying matching IDs as well.**
   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. **I believed there were sections of my code that were clean and precise in what was needed. My Task class and Appointment class were right where I wanted them to be. Below you can view the “getTaskID” method as an example.**
         1. public final String getTaskID() {return taskID;}
         2. public final String getName() {return name;}
         3. public void setName(String name) {
         4. if (name==null || name.length()>20) {
         5. throw new IllegalArgumentException("Task name is invalid. Task name must not be empy and may not exceed 20 characters.");
         6. }else {
         7. this.name=name;
         8. }
         9. }
      2. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate. **I made sure my code was efficient because I focused on being brief and precise with my methods. I wanted to execute in a way that accomplished what I needed to without being overly “wordy” with my code. I feel that refactoring and making sure there isn’t more room for error is the smarter way to go about it. Please refer to my example below as a block of efficient code:**
         1. private void checkTaskID(String taskID) {
         2. if (taskID==null || taskID.length()>10) {
         3. throw new IllegalArgumentException("Task ID invalid. The Task ID must not be empty and may not exceed 10 characters.");
         4. }else{
         5. this.taskID=taskID;
         6. }
         7. }
2. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details. **I have been focusing on black-box testing primarily. I believe that is a great technique for the requirements we must deliver for this Project. We are testing specific inputs, so I also have been utilizing equivalence partitions and set parameters that must be met for the inputs to be valid. Again, I believe this is the most efficient way to test the program requirements as we have user input and set the bounds, so the user does not submit an invalid input. This creates integrity in our application and allows us to have an expected outcome.**
      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 white-box testing or experience-based testing. I did not create any tests around the idea of exploratory testing, error guessing, or checklist-based testing. I feel that these tests would not produce the best results since they are specifically minor input fields. If I was developing something a little more complex or had specific inputs and could think of how a user would commonly make a mistake, I would have used these techniques. As for the white-box testing, I feel that this option is also a little too complex for what the requirements are asking. We are not developing this application on a massive scale, and I feel that white-box or more designed for larger scale tests, at least for statement testing. We could argue that decision testing could be beneficial because we do deal with IF-ELSE statements in our tests. If the characters exceed a certain limit, we throw an error. This may be an exception, but for the majority, we focus on black-box testing and equivalence partitions.**
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations. **If we were building a game, we would utilize more white-box testing with decision and statement testing because the scale is much larger and we are testing through different levels of a gem/environment. I would then leverage experience-based testing here as well to ensure we are trying to anticipate what our clients will do/run into with a game. On the flipside, I would assume black-box testing to be more specific and structured, like the input of an online form for a company or standard websites. I would also think that apps that make reservations or search for things would be more black-box based. To the best of my ability, this is how I would choose a testing technique for this situation.**
   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. **I tried to be very cautious when working on this project. There were multiple variables and I wanted to ensure that I was crossing all my t’s and dotting all my I’s. I feel that it was important to appreciate the complexities of the code because it was important to understand what was happening and what needed to happen with the program. Following the requirement list for each class was pertinent to what we were trying to accomplish. I also believe that understanding the relationships between each class and test was important because without that basic understanding, being able to get our 80% coverage would’ve been challenging.**
      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 feel that I tried to step away from knowing I had a hand in this code and tried looking at it through a fresh lens. I find that when working on your own code, you may overlook some functionality or process if you are the only one creating, running, and testing the code. I always believe that an outside source should look over your code to ensure you are looking at the code base from all angles. I think that bias would be a concern if I was responsible for my own code because you are very close to the project. Especially if you’ve been building and running into problems in that area, you must ensure that you are disciplined enough to be critical of your own work. You also must have the expectation of analyzing your own work, even when you are so close to it. I believe that with proper experience and training, one is able to limit bias and properly test their own code.**
      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. **I feel that there are many reasons why it is important not to cut corners when it comes to writing or testing code. One reason I feel it is important is because as a software engineer, you want to provide solutions, not more problems. The integrity of your work is known throughout your teams and it’s important to think of all situations your code/program could be used in. I plan to work closely with my team to ensure we are following the appropriate techniques with white-box, black-box, and experience-based testing. I would also like to work in a team that is thorough in planning every detail of the requirements so we can ensure that we have all our bases covered from inception.**