Nicholas Wyrwas

CS-320-11359-M01

Mrs. Parul Hirpara

08/18/2024

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.
      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.

Project One involved creating six classes to meet the software requirements for the client. For each variable in each class, my app designed for a specific purpose exactly as per the project specifications. In order to meet all the requirements, the variable had strict checks to make sure that all inputs were as required. For example, the variable ‘contactID’ in the Contact class was specified that it should not be longer than 10 characters, while not being null. For this part, I forgot to add this specification but when I realized my mistake I went back to the source code to ensure that this was added. To check the validity of the input handling, I added JUnit tests for all of the Client, Task and the Appointment classes. While the first kind of test was used to check if the classes would accept valid inputs without any errors. For example in the Contact class, I provided inputs of a valid user ID, first name, last name, phone number and address. It was required that the class would process it without any errors and that the getter methods would work accordingly. The second kind of testing done was to provide inputs which would be considered not acceptable in the criteria. For example, in case of the phone number field, it was required that the number should not be more than 10 digits. It was checked if the error was populated. If the error was not built, then the test would fail. I maintained coverage, by ensuring that the test coverage would maintain a percentage of 80% or higher.

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

Since my JUnit tests are written in Java, I also added comments to make the code more readable. For example, in the taskServiceTest.java class, when online we can see a public void testAddTask() method which the comment tells us is used for testing to add a task. I did this commenting so if I wanted to check if everything is going accordingly and written correctly, and it also helps me to find my errors in the code easily since I know what each section is about.

Furthermore, I also took care of formatting and naming considerations according to best rules, which also made the code clear and easy to maintain. Besides the JUnit tests, I also made sure to do static and dynamic testing of the code, and I consistently added comments to make the code not only efficient but also very readable.

A screen shot of a computer program

Description automatically generated

1. Reflection
   1. Testing Techniques
      1. What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.
      2. What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details
      3. For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.

I used two types of software testing techniques which were static and dynamic testing followed by JUnit testing. The first static and dynamic testing are two different ways to test the code. Static testing is done without executing the code. I have to review it line by line to check the syntax, structure and logic. If I find a syntax error or any other violations of coding standards, or if the code has a logical flaw, I will report it. Like static testing, this practice is important since it helps to catch any defects before executing the code, which can potentially minimize errors created during the execution of the code. But dynamic testing entails running the code in a controlled environment and seeing if it behaves properly. This stage of testing is about verifying that the code behaves the way it’s supposed to when it is executed. So, by running the code with a bunch of inputs, I could verify its correctness, check for runtime errors, and observe its behavior under varying environmental settings. The combination of static and dynamic testing ensured that the code was bug free and functioned as intended. The second test I did was the JUnit test. The JUnit test was separated from the code as a distinct JUnit testing file. I wrote tests with the expectation of providing it with some data into the class and corresponding output, to ensure that the output matched the expected output. In this way, the static and dynamic testing were complementary, allowing for thorough validation. This validation included testing whether or not the software met the requirements and produced the expected outputs in all situations.

* 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.
     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.
     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.

A screen shot of a computer program

Description automatically generatedAs a software tester, I essentially exercised a considerable amount of caution throughout the testing process. Where I wanted to be fully aware that if any tests or attention to detail could result in any overlooked errors within the code. Where in an actual industry any overlooked code could produce vulnerabilities or even be exploited. So I wanted to ensure that I was able to not overlook anything and understand everything that was needed to be completed to make sure the program functioned as needed. When evaluating the JUnit tests, it was important to review any of the code that was highlighted in red. Knowing that it failed or was not called upon. For example:

And with noticing that some parts of the code were not being highlighted green, would essentially let me know that the overall test coverage was being impacted. Lowering my coverage percentages. While also showing me what I need to focus on reviewing and understanding why that part of the coverage test failed. To minimize bias during code reviews, I would essentially take breaks when working on or reviewing my code. This would allow me to come back to the code base with a fresh perspective, giving me a clear mind when reviewing it again. Which would allow less bias to occur to an author of the code. When it comes to maintaining any discipline as a software engineering professional, it is utmost that we remain professional at all times. When a software engineer wants to cut corners it actually can hurt a company or team more than help. As cutting corners when testing can cause someone to overlook something and allow errors to be released into production. Which can cause vulnerabilities to allow hackers or other individuals to exploit any system bugs. When it comes to avoiding technical debt, I want to commit to understanding software requirements more by thoroughly and properly. Where I will provide proper testing while following all industry best practices. May it be a software engineer programming a new software or program as well as testing.