Project Two

Nick Allen

October 14th, 2023

CS-320: Software Test and Automation

**Summary**

**To what extent was your testing approach aligned to the software requirements?**

I aligned my testing with the software requirements by making sure each requirement was met with at least one JUnit test case. For example, to test for uniqueness, I added ‘testAddContact\_UniqueID()’ which tested two contacts with the same ID and made sure the second would throw an exception.

**Defend the overall quality of your JUnit tests. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**

My quality appears to be high based on my coverage percentage being 83.2% according to Jacoco. I’ve also covered the test cases specified in the instructions for the assignments, which means that each of the tests were meaningful, and not just made to create high code coverage.

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

For soundness, an example I can make is that for the ContactTest class, I included tests checking for null values and string lengths to be sure the contact object is valid upon creation. Another example is in TaskServiceTest, I included tests to validate the tasks should they be added, updated, or deleted.  
**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

For efficiency, I used a hash map for quick look-up times in the TaskService and ContactService classes. This makes it to where the ID is used as a key to quickly look up or modify the information we need, making for better performance.

**Reflection**

**Testing Techniques:**

**What were the software testing techniques that you employed for the project? Describe their characteristics using specific details.**

1. Unit testing: I used Junit to make isolated test cases that test individual functionalities of the program, such as adding, deleting, and updating.
2. Boundary value testing: Each service has upper and lower boundary limits that were tested.
3. Exception testing: each service has tests for undesirable conditions, like adding null values.

**What are the other software testing techniques that you did not use for the project?**

1. Integration testing: This program was not added to a larger system to check how it will interact with things like UI’s.
2. Stress testing: we didn’t overload the system to test how it handles and measure failure. E.g. Adding thousands of appointments at a time, etc.
3. Performance tests: Didn’t include any tests to see how the program responds under a heavy workload.

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

**Unit Testing:** the practical use in unit testing is that it definitely eases debugging, but at the cost of taking more time to write tests alongside code. Worth it in my opinion, I’d rather do the work on the front end than try to fix problems after I finish writing the program.

**Boundary value testing:** The practical use is that it helps us maintain our requirements by testing the boundaries of what we need to be useful input from the user. The drawback is that it will catch edge case errors but might not catch errors inside the boundaries.

**Exception testing:** can catch unpredictable errors by the user but take a ton of extra time to setup as each exception can be handled differently.

**Integration testing:** This testing will help integrate the program into a larger infrastructure, but this means catching bugs will be more complicated as there are more moving parts.

**Stress Testing:** Its practically used where the software we make will be used by large volumes of people and can help determine where the cost to performance ratio lies for a project. The drawback is that if you find critical errors, fixing them at this point is very expensive.

**Mindset:**

**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?**

I employed caution to a higher extent, making sure to be more exact in my tests than I would otherwise be developing software. At this point in the process, missing issues to test may cause them to be more expensive to fix down the line. If you don’t take the complexity of the software and how it relates to other code into account, it’s very unlikely you will catch errors that arise from them interacting with each other.

**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?**

I tried to limit bias in my code by implementing feedback from someone much more experienced (Dr. Krava) without arguing a different viewpoint I think may be better. If I were responsible for testing my own code, it would be very easy to override or forget what may be great feedback simply because I created the code. This could lead to more serious issues down the road so it’s important to let the ego take a backseat, especially if you’re testing your own code.

**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?**

Cutting corners usually causes more issues than it solves. If you do things right the first time, like detecting errors early documenting and reviewing your code with others, etc. it will avoid most of the technical debt that occurs when you push out a program that has too many corners cut for the sake of speed (or laziness). My plan for avoidance of this technical debt is to pay close attention to detail and follow best practices when both creating programs and testing them.