Matt Zindler

Professor Walker

CS-320

11 December 2021

Grand Strand Systems Reflection

The unit tests I designed for my code were aligned with the requirements of my code. Many of my tests were focused on making sure that the program would be able to add, create, and alter the different classes and methods that were present. For example, in the ContactServiceTest class, there are tests to ensure that check for unique IDs in different instantiations (code lines 22-35 ContactServiceTest.java). Tests like these ensure that my program can fulfil requirements set by the assignment. Test coverage also helps me to ensure tests can support requirements by letting me know which methods are tested and which aren't. My coverage could have been higher for this project, as I didn't test all my set and get methods, which lowered my coverage considerably. In a future update, I would account for these in the tests. At the time, I didn't think I needed to write tests to cover them, but decisions like that can lead to errors slipping by.

The JUnit tests helped me write better code and spot errors much easier than rereading my code and trying to figure out what was wrong. As a specific example, I was writing my tests for the first milestone, and realized I had errors in calling the class, which the test was able to point out. In addition, each test class had tests to ensure each linked class could instantiate the class to begin with. Both these types of tests would point out simple errors in the code that I could fix. Efficient code is another important aspect for coding, mine was made efficient by having each class be unique and not writing duplicate code. A more specific example is in the

Task class, I integrated exception throws for invalid data coming into a new Task class as opposed to more complicated if else statements (lines 15-28 Task.java).

The two main types of testing I used for this project was unit testing and acceptance testing. Most of the testing was done with JUnit tests, to ensure that the software was able to pass simple tests that were expected of it, like creating classes and verifying unique IDs. Acceptance testing was the other type of testing I implemented for this project. Acceptance testing means that the program is able to meet the requirements set by the assignments. This was important for the project so that I would actually get credit for the program being able to work as intended. JUnit testing also helped in this way by showing that the requirements set would actually be hit when the program would run. I did not use the other testing types for these milestones, as they are more involved in projects that are going past the development stage. Integration, system, and performance testing all happen further into the SDLC than I was at for these projects. Integration and system testing would take place when the different modules are coming together, along with systems such as user interfaces. These two testing types ensure that different modules play well together, and the systems work well together without issues. As for performance testing, I felt that I did not need to use it as we were not building fully functional projects for these milestones, and code would not be run that was not JUnit tests.

These different testing techniques each have their place in the SDLC. Unit testing generally takes place first, and lets the developers know that the program/classes functions on a base level as expected. Integration and system testing are implemented after the development step, and is used to test the program as a whole. These tests work to find issues with classes and calls between functions work as intended (integration), and tests that the software works together as a whole program (systems). Acceptance testing takes place near the end of the SDLC, and

helps ensure that requirements set by the customer are met. Functions and features should be working as intended, and are usable by an end user. Performance testing takes place around the same stage, where the program's performance on intended hardware is tested. This is an essential test, as you need to ensure that the program is able to run smoothly, and is compatible with the customer's hardware.

The mindset I adopted for this program was to write most everything I could, as per the assignment requirements, and then create some tests to check their functionality. After the tests were created I was able to verify what worked and what didn't for my tests and my project. The main issue with this was if I had an issue, it would be hard to tell if the error was coming from the tests or the main program. In the first module I had a bit of an issue with this, as I had forgotten to add the @TEST for the unique test, and thought my issue was in the main code. This became less of an issue later on, as I was able to refine my tests, and build off my first module structures. Bias can be an issue in testing by being gentle with your tests, or hardcoding in answers. I tried to limit bias in this way by writing tests I wanted to pass, not only the ones I could. In the future as well, I would include more tests for get and set methods just to be thorough. Being disciplined is an essential aspect of being a good coder and writing good code. Being able to do things the right way, and setting up foundations for others to work on makes a better product for the customer, and makes others jobs easier, whether it's for your team or for others in the future. Whenever I think about leaving out comments or writing unreadable/copy pasted code, I always try to imagine what it would look like for someone else to read and fix what I wrote. To avoid technical debt, I plan to always document what I create, as well as to code the right way rather than taking shortcuts like hard coding, and leaving out important parts of a project. Examples in my code are the comments I left, whitespace for readability, as well as not

hard coding any of my classes or tests. Another specific example was using error handling for invalid inputs, as opposed to having a long and complicated/hard to read if/else statement to cover it.