Cody Faircloth

Dr. Ramsey Kraya

CS 320

April 13, 2023

Project Two: Reflection and Summary

My unit testing approach was directly aligned with the software requirements. Each of my non-service test files is a testament to this approach. I will be using AppointmentTest.java for evidence but this approach is true for each file. In my testing, I understood the requirements and tested each function and variable in the class to ensure they followed the requirements. For example, in AppointmentTest.java lines 37-43 test the accuracy of the creation of an Appointment object. Lines 45-73 include three separate tests which check the constructor to throw errors when a variable does not meet the criteria, whether it is too long or null. Lines 75-99 check each mutator in two tests check for accuracy and errors are thrown when a variable does not meet requirements. Overall, my JUnit tests appear to be of high quality. Each test file had a test coverage of 80% or higher on the source code that is being tested. For example, AppointmentTest.java boasts 100% code coverage on the Appointment.java file. This trend follows throughout the project with each test file achieving the standard 80% coverage or higher.

I ensured that my code was technically sound by following standard best practices for developing in Java. For example, each source file has a constructor, accessors, and mutators when present. This is shown throughout the entire Appointment.java file. The constructor spans lines 8-21, the accessors span lines 23-31, and the mutators span lines 33-45. Additionally implementing IllegalArgumentExceptions kept the code technically sound regarding requirements. If an argument is passed in the creation of an object or the mutation of a variable

for an object, an exception will be thrown. This eliminates the possibility of requirements being impeded. Efficiency was an aspect of this project that increased over time. In the first couple of assignments, my code was inefficient, I included much redundant code that could have been simplified with the implementation of additional methods in the source code and @BeforeEach statements in the test code. ContactTest.java lines 9-24 show the trend of inefficient code since the same constructor and variables are being used and coded many times over, leading to redundant code. However, in AppointmentTest.java lines 23-35, I addressed this inefficiency and implemented a @BeforeEach statement to make my code more efficient. Further efficiency could be reached by adding additional methods to the source code to check for requirements related to each variable. However, this was never implemented into the project but upon reflection, I recognize this deficiency and will keep this in mind for future projects.

During this project, I implemented black-box testing and white-box testing. Black-box testing is testing code without prior knowledge of how the software should function. In this project, this was primarily used to ensure all software requirements were met. An example of this type of testing presented in my project would be the tests that occur in AppointmentTest.java. These tests solely test requirements rather than functionality. White-box testing involves testing that requires knowledge of how the software should function such as interactions between various data types. This testing technique was primarily used to check interactions between the non-service and service classes such as creating objects from an array and deleting objects from an array. Examples of this type of testing can be found in AppointmentServiceTest.java where appointment objects are created, added to a list of objects, and deleted from the list of objects. In this project, I did not perform any non-functional testing. Non-functional testing involves testing the non-code-related aspects of software such as security, performance, and reliability. It was not

practical to perform this type of testing since this project was not being implemented into a greater system. Each of these testing techniques has practical uses, however. Black-box testing is beneficial to understand and implement requirements for your software. White-box testing ensures that your software is structurally sound and that interactions between various aspects of your code function correctly. Non-functional testing is imperative to implementing software into a larger system. When a project's scale and complexity are larger, non-functional testing is necessary. Every type of software testing technique has practical uses and should be implemented when practical.

The mindset I adopted when working on this project was my default mindset any time I code. I set my goal to meet all requirements presented, create secure and quality code, and deliver a functional piece of software. Acting as a software tester I do not feel like I employed much caution. I tested every file as thoroughly as necessary to ensure functionality. Since this was a simple project, I do not believe much caution is necessary; however, more complex projects should employ a higher level of caution. It was important to appreciate the complexity and interrelationships of the code because these aspects build a framework for the tests you will run. For example, in this project, it was important to understand the complexity of requirements for each variable so these variables could be tested to ensure the requirements were met. Additionally, it was important to understand the dependencies of classes such as the dependency of the AppointmentService class since appointment objects are created and added to an array in the AppointmentService class. I did not need to mitigate bias in this scenario since I always strive to better myself as a developer and a tester. I wanted to make sure that my code worked correctly, and I wanted to strenuously test my code to ensure my code was of the best possible quality for this project. I did not see bias being a concern while testing my own code; however, I

could see this being an issue in the future. I see myself as a learner and an intermediate-level developer meaning that I am always up for self-criticism and self-improvement. I could see the frustration of higher-level developers and the bias that comes along with it while testing their own code. As I continue my software development journey, I plan to keep myself humble and always be open to criticism whether it comes from myself or others because the important part of any skill is developing it over time. Being disciplined is an important aspect of any hobby, skill, or profession. Cutting corners should never be the route taken in any endeavor. When you cut corners while writing or testing code as a software engineering professional, you leave your software open for vulnerabilities and exploitation which could hurt your organization's reputation and your professional reputation. Additionally, this practice could leave your customer base vulnerable. Personally, end-user/customer security is the most important aspect of software development and I plan to carry that value throughout my academic and professional career. I intend to avoid technical debt by doing things right the first time and testing frequently and thoroughly.