**Philip Shaw**

**CS 320**

**10 December 2023**

**Project Two (Module Seven)**

**Summary**

Throughout the creation and testing of this course’s project, my approach was largely consistent. The primary goal was to center methods and attributes around “Grand Strand Systems’” individual requirements as directly as possible. Although requirements can rarely be translated into code verbatim, adhering to the prerequisites as closely as possible invited less room for miscommunication and feature creep. Unit testing for the primary application classes largely focused on creating negative tests to assess the boundaries of input. For secondary “service” classes, tests were mostly positive, confirming that the primary application classes could be utilized as intended. In addition to returning satisfactory deliverables, another reason for attempting to align my test cases “one-to-one” with the requirements was to ensure the *test coverage* percentages reflected the *requirement coverage* as accurately as possible. A high test coverage percentage is only a convincing achievement if each test method is relevant. I feel comfortable knowing that the 92% coverage yielded by my tests are based on a relevant scope of input and object instantiation.

As this project provided my first experience with JUnit and writing unit tests in general, the requirements provided a helpful framework for me to work from. I believe that attempting to adhere to these requirements helped me to expedite my core knowledge on unit testing, and to remain undistracted by the wealth of features provided in JUnit 5. Of course, attempting to meet requirements without added interpretation has both benefits and shortcomings. Working this way gives clients what they’re asking for, but possibly not what they’re trying to achieve. For example, the delivered Appointment class allows appointment objects to be created with *descriptions* and *dates*. However, there was no request for a *time* attribute. Although I did consider including this feature, I ultimately opted to exclude it from the deliverables due to questions that needed addressing first. These included, “Which time zone does Grand Strand Systems operate in?”, “Could appointment times overlap each other?”, “Which hours of the day would appointment bookings be limited to?” Despite this exclusion, I’m satisfied with the result of the project. This is primarily thanks to guiding feedback I received from Professor Tuft. His insight helped me to improve the efficiency and soundness of my code. For instance, I was providing static “future” dates within my tests, not considering that these dates would need to be updated once they had passed—in order for the test classes to pass without being refactored. I was advised to use dynamic dates instead, such as: LocalDate.now().plusDays(“1”). Overall, working with JUnit was a rewarding and eye-opening experience. I’ll continue to use this framework for future Java projects, as it’s much more powerful than using print statements to test code.

**Reflection**

Testing performed throughout this project was primarily functional. The scope focused on three modular components—Contacts, Tasks and Appointments—that collectively formed the larger application that was developed for Grand Strand Systems. These tests were all performed manually, initially beginning with print statements before being transitioned to JUnit classes with the potential for automation. Testing the first “Contact” module began as an “ad hoc” process due to my inexperience with writing unit tests. At the time, my sole objective was to ensure the code actually behaved according to expectations—based on a reasonable breadth of input. It’s worth noting that a majority of this testing centered around boundary values for respective string input. This strategy remained consistent for Contact, Task and Appointment class objects.

Other testing techniques I considered employing were integration testing—for the complete application—and automated testing. Although I’m content with the resulting product, it’s regrettable that I wasn’t able to learn how these techniques are employed, as they’re pillars of the DevOps methodology (Erich et al., 2014). However, the learning curve for meaningfully utilizing these techniques is likely too steep to cover within an 8-week course/development period. These testing concepts seem fundamental for implementing continuous integration and for automating testing when code is committed to repositories.

Not only did/would these testing techniques collectively prove useful for Grand Strand Systems’ application, but for many of the popular applications used by billions of people worldwide. These include social media platforms, ridesharing apps, financial programs, video streaming services and many more. The aforementioned software testing techniques facilitate the global scale of development required for these applications to survive. Within DevOps frameworks, these testing techniques support growth for automation, fluid integration of updates and the ability to rollback changes as needed. This remains true on both holistical and atomic levels.

**Mindset**

Over the course of creating an application and using JUnit to test its methods and classes, I developed a much greater appreciation for the process of gathering feedback. This includes feedback from experienced developers, end users, and everyone in between. I came into this project with a subconscious notion that solitary development is best suited to my workflow, because it involves fewer moving pieces. However, it’s incredibly easy to forget that a singular perspective is not usually the *whole* perspective. This is a chief reason why there needs to be a division between development and testing. Using a previous example, I was entering static dates while testing input for the Appointment class. I hadn’t considered that these dates would no longer provide valid results once they had passed—one of the very requirements I coded into the class. That meant my tests would only pass until those dates expired. Here, Professor Tuft’s feedback helped me to solve an issue that I wouldn’t have even considered alone.

Additionally, I approached this project with some care because hypothetical user data would be stored in the application. However, I didn't exercise the formalities required for much more sensitive applications—healthcare, military, finance, etc. There is an argument to be made that personal data should be handled with equal sensitivity. One piece of advice I received from Professor Tuft helped me to encapsulate personal data more securely within my application, better fortifying data for its hypothetical users. Of course, when coding an example project, it’s easy to take for granted that “real” users won’t be at risk if important security details are overlooked. While this might be a short-sided sentiment, it’s difficult to simulate the importance and gravity of protecting real user privacy. However, this is a necessity for anyone planning to develop and test software for real users in the future. Commiting good practices to habit allows developers to approach sensitive situations in a more structured manner.

The most important thing I learned throughout this project is that understanding how to elicit and adjust to feedback is the most core principle of development. This includes feedback from peers, clients, customers, and end users. Other aspects of development can be outsourced or automated far easier than this aspect. Understanding the nuances of feedback helps to differentiate between what is being asked for and what’s needed. It can also prevent our own biases from getting in the way of misinterpreting these differences. Lastly, it was beneficial to actively remember that “going the extra mile” doesn’t just mean performing the motions of doing a good job, but that this effort provides real benefits for other individuals and organizations.

Reference:

Erich, F., Amrit, C., & Daneva, M. (2014). Report: Devops literature review. University of Twente, Tech. Rep. Retrieved from: <https://www.researchgate.net/profile/Chintan-Amrit/publication/267330992_Report_DevOps_Literature_Review/links/544ba33f0cf2bcc9b1d6bd8a/Report-DevOps-Literature-Review.pdf>