Christopher Fusaro

February 22, 2025

7-2 Project Two

CS-320-12209-M01 Software Test, Automation QA 2025 C-1 (Jan - Mar)

During the development process for project one, I noticed the development of each class and its associated tests. The first approach I took was to create the class, as well as the attributes and the methods. This included the constraints to accommodate the supplied requirements per the customer. This was definitely the section that took me the longest to develop. This was due in part to my lack of understanding of JUnit and also my first ever attempt at writing tests. The second part, designing the task class, task services, and their accompanying test, went much quicker, partly due to focusing on the tests first and then developing the class around the tests. The Contact Class and ContactService were by far the best of all modules. I implemented the Don’t Repeat Yourself methodology and was able to keep the constraint in its appropriate methods, which allowed for less duplication at the class level. The coverage report became my main focus and I found that I was testing for everything that was unacceptable. So I created tests that would test values that were acceptable, which increased my coverage. I believe I was able to cover all the software requirements and had adequate testing to cover it. My JUnit tests were good quality, and my lowest coverage percentage in the classes was 88%.

Writing the JUnit test was very straightforward and easy to read. I felt that this was the area that became extremely bloated with duplicated code. I had to initialize an object and give it some default values so I would have something to test against. So I did some research and discovered fixtures and was able to use the BeforeEach portion of the JUnit model to initialize the same object with the same default values before each test, which reduced my total lines of code by a significant amount. This was also a benefit in transcribing the object into a previous test. I had made a typo and couldn’t figure out why I couldn't pass my test. This had the added benefit of increasing the efficiency of the tests due to the objects being set up ahead of time.

In the appointmentTest file, I tested that the object was able to be created with acceptable values using the assertTrue method on the supplied attributes after creation. This was a good confirmation that the constraints were passed and the object existed(lines 26 -30). Proceeding the test to create an appointment object, I tested each attribute with a test for each constraint. The way I tested each of these constraints was by testing if an IllegalArgumentException was thrown. Lines 34 - 38 tested if an object's ID could be set to null and an error was thrown.

The efficiency of the object and the test was not as good as I would have liked, and the reason for this was that the object was made very secure with the attributes being set to private, along with all the setter methods being private. This made the object very secure but more closed off to testing. The only way to access the setter methods was through creating the object. So each test involved the creation of a new appointment that was expected to fail due to running up against a particular constraint.

The software testing techniques employed were automated tests, consisting of unit tests, integration tests, and functional tests. An argument can be made for security tests as well because some of the data fields were limited to integers, which would prevent malicious code from being added to that field. Each individual class was tested using a unit test to ensure they followed the requirements. Each of the services that were tested involved functional and integration testing. The TaskService class was tested to add a single task and verify that it was in the list(TaskServiceTest lines 24- 30). Next, a test was made to add three separate tasks and confirm their presence(lines 35 - 49). This was a functional test because it was able to employ multiple steps that moved across to classes in a way that would function in the wild. This would also fall under integration testing as it tests the interaction between different parts of the system, ie task and taskService. These tests focused on the connections and dependencies of these objects.

System testing, performance testing, User acceptance testing, and end-to-end testing. These were the types of automated tests that were not used in the testing of this software. The system testing was not used due to a lack of a whole system. User acceptance testing is a verification that all requirements identified in the requirements gathering phase are met. This is typically performed by end-users, business stakeholders, and product owners. Performance testing is a measure of how long a system takes to respond to a user's request. This is used to identify low performance pieces of code. Instabilities that may crash a system under heavy load. Also, any bottle necks that cause low throughput. End-to-end testing is used, similar to system testing, to simulate user interactions through an application from start to finish. This is important for quality assurance and detecting flow state errors that may only occur during user interactions.

Throughout this project, I saw myself as a developer who was trying to harness testing to increase my productivity, software resilience, and agility to try new things while maintaining the requirements that were set forth. This is the type of career that I am aspiring to. So it made things more real for me and pushed me to try new things and explore more than I may have if I was just portraying the role of a tester. There is a question of to what extent did you employ caution? I am not really sure how to answer that. As a software tester, I think it made me very cautious to where I added extra tests to ensure I was adequately capturing the requirements that had been conveyed. But this caution was what allowed me to take risks with developing the class with unfamiliar data structures and more complex methods. I knew if my tests were good, I would be free to take greater liberties because I could count on guides not to let me get too far off the beaten path.

Bias is a serious concern in software development, and the way I tried to combat it was by sticking to the requirements as they were precisely laid out and not trying to interpret the direction any deeper than what was explicitly stated. I am always brought back to the development of facial recognition software and how the bias was introduced by the training data being of primarily caucasian faces. Then the realization that the system that they worked so hard to build was stifled by an oversight. This is where the product owner may be a valuable resource who can provide insight into a product and help keep you from introducing your personal bias.

Finally, discipline plays an enormous part in developing professional quality software. If there are two paths forward, the path chosen should be taken because it is the right path, not because it is easier or because it will be more costly to drive up the price. It should be taken because it is the path that leads to the best possible outcome and provides the most value to the stakeholders. It is essential not to cut corners in writing or testing code because in the long run you may cause sizeable refactors to need to take place. After all, the testing was too broad or the constraints were not narrow enough. This could make the entire project fall short and be undeliverable. Technical debt is a severe symptom of not correctly gauging the requirements of a project that can plague a developer for a long time. Strategies for minimizing technical debt can be using coding standards and design patterns. Use a well-structured class hierarchy when using an object-oriented language. Test driven development offers a testable and well defined path forward. Continuous Integration and Continuous Deployment allow for automation to catch issues earlier and can check to ensure code meets specific standards before being committed. Using a static code analysis tool can also help root out potential bugs and security vulnerabilities.

Schmitt, J. (2024, January 17). *Test-Driven Development (TDD) explained*. CircleCI. https://circleci.com/blog/test-driven-development-tdd/

*What is Ci/CD?*. Red Hat - We make open source technologies for the enterprise. (n.d.). https://www.redhat.com/en/topics/devops/what-is-ci-cd

What is SDLC? - software development lifecycle explained - AWS. (n.d.). https://aws.amazon.com/what-is/sdlc/