Grand Strand Systems Summary and Reflections Report

Summer Bernotas

Southern New Hampshire University

CS-320: Software Test, Automation, and Quality Assurance

Professor Norman

23 June 2024

Summary Report

After running Junit 5 Jupiter testing, I was able to successfully run unit test on all sections of the application. Contact Service, Task Service, and Appointment Service were specifically developed inline with the given requirements. I ensured that each class had a functionality that coincided with what was expected out of that class. The Contact Service application specifically required a way to store a first name, last name, address, phone number, and an ID under a contact. These variables had their own requirements of maximum character length as well as the ability to not be null. The ID needed to specifically be unique and unchangeable. We then were expected to ensure a way to update each variable, excluding the ID, as well as add a contact and delete a contact. In the Task and Appointment Service application, the requirements were quite similar. We needed a way to store a task and appointment name as well as a task and appointment description, each with their own maximum character length and no ability to be null. We then also needed a unique and unchangeable ID for the task and appointment which could be stored in a list that can be updated, added, or deleted. The appointment application specifically required the use of a date so that the appointment could be set for a date that is not null or in the past. I made sure to code each required functionality as well as catches to ensure that the required length amounts, etc. were handled properly. To ensure a nonchargeable and unique ID, I utilized the AtomicInteger function within Java that was recommended to me as well as utilized a HashMap to efficiently store the task, appointment, or contact itself.

Although I know there could be some improvements, I would say that the quality of my JUnit tests was high. I ensured that there was only one specific way that a required functionality could be done. For instance, updating the task description could only be done by

accessing the task through its ID and calling the Task.java function to specifically update the task description. This, just like all other variables, is the only way to directly change that variable without directly changing the code. This allows me to ensure a well-written and accurate unit test as I perform that specific function and see the outcome. Each test performs a very specific action to ensure that only the desired area is being tested and it cannot be thrown off with anything else. I would argue that my code is both technically sound and efficient as it utilizes basic coding principles as well as being coded with functionality in mind. The use of HashMap's, final variables, etc. all were used with specific areas in mind to reach requirements.

Reflection Report

For each application, I utilized only unit testing. I decided to use unit testing because they are fast, easy to write, and what I believed to be the best option for these specific assignments. They allowed for tests to be repeatable which was needed for testing null input as well as input length. They were also able to help me catch bugs sooner rather than later because I could implement a unit test as soon as I was done writing a specific section of code, which overall, prevented future errors that may have derived from that code if it was not caught beforehand. For instance, one of my unit tests caught an error with the Task.java class that was incorrectly setting one of its variables to a different variable. Although this was a very simple fix on my end, it did allow me to catch it before I got more in-depth with the code where this could have easily been missed.

I did not use any other form of testing because I felt that it was unnecessary for these assignments and for the stage that they were in. I decided against integration testing, even though it is foolproof, because it has a much stricter way of execution as compared to unit testing. I felt

that these systems were not too complex yet where I needed to do such in-depth testing that followed specific orders. I also decided against system testing because I wanted to test the classes and inputs specifically aside from the functionality of the application itself. As the application did not have any user interface, I just felt this was not needed. I also was able to keep track of system requirements in another way which made this testing unneeded.

Unit testing is a good choice for initial testing of functionality, especially those involving input, output, or logic-based functionality. It ensures that test cases can be written for all functions and methods so that when a change takes place within the code, it can be identified quickly. Integration testing on the other hand is very important overall and is commonly mandatory. This testing checks the overall functionally of an entire system, ensuring that all components are working together correctly. Unlike unit tests, integration tests have a strict order of execution to follow. Lastly, system testing verifies that a systems functionality is meeting its requirements. This prevents defects and missing specifications to ensure that the system is meeting all intended functionality.

Reflection on Mindset

Creating unit tests for my own code definitely changed my mindset on coding itself. It has now gotten me to see how testing works first-hand, allowing me to be more aware when writing code. I can pay mind to what type of tests may be run and structure my code accordingly which would hopefully lead to less error. It is vital to appreciate the complexity and interrelationships of code that I was testing to ensure everything aligned with each other and did not cause error between each section.

To limit bias in code review, I made sure that the tests being run were as accurate as possible. I did not change my test or code to fit what I needed to make the test successful. I wrote

unit tests to take input exactly as they were expected and if errors were found then I changed the code accordingly in order to properly fix them. For instance, the appointment name and description could only be changed one way. That is to be accessed from the TaskService class and utilizing the update methods which took in the tasks ID. I wrote the unit test to do exactly that – find the task with its ID and update the specified section of the task. If the unit test failed, I did change the test to ensure it would work as that would be biased of my own work and denial to accept that I made an error. Instead, I looked over my code and fixed any mistakes that I made to ensure the methods worked as they were expected. Being able to accept that you may not always be perfect and that making mistakes is okay – as long as their fixed especially when looking at life-or-death applications (aerospace software, etc.)– is vital to most aspects of life, even when coding. You do want to ensure that you are writing to the best of ability to provide to most high-quality code possible. Doing so will lead to less error and more overall success.