**Tammy Hartline**

**06/16/2023**

**CS-320 SNHU**

**Final Project- Summary & Reflection**

**SUMMARY**

I used JUnit to write unit tests that ensured the functionality of each of my programs was as efficient as possible. For the Contact, ContactTest, ContactService, and ContactServiceTest files, I tested a variety of inputs and outputs to verify that my program produced the correct output in each test case. For the Task, TaskTest, TaskService, and TaskServiceTest files, I concentrated my testing more towards verifying the program handled errors and threw the correct exceptions for invalid inputs. The final files, Appointment, AppointmentTest, AppointmentService, and AppointmentServiceTest, were written to test the correctness, so to speak, for each algorithm in the test cases. To accomplish each of these tests and ensure that I met the 80% total coverage requirement, I was able to combine each of my unit tests into a test suite, that executed all tests simultaneously and produced better than expected coverage of 100%. Doing this ensured my JUnit tests were of excellent quality with very effective and efficient coverage.

To ensure my code met all the software requirements I continuously referenced back to the rubric given and included each of the requirements in my code comments as a reminder of the constraints that needed to be written into each method, and guide for each of my test cases. I also tried to ensure that each of my tests were able to function on their own, on a per code block basis, or as a group when ran as a whole, by consistently running each method each time a change was made to the program. I used standard yet descriptive naming conventions for all methods and classes created, to ensure my code was readable and as easy to understand as possible. Lastly, I checked the coverage percentage and addressed any and all gaps in coverage, until I achieved 100% coverage for every file. To further demonstrate the effectiveness of the processes I used, I have included several screenshots of my code below:

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

**Reflection**

**PART 1 – Testing Techniques**

For this project as a whole, (Contact, Task, and Appointment), I used several testing techniques learned throughout this course. Some of these testing techniques include unit, acceptance, integration, and some regression testing. I relied heavily on unit testing, in particularly for the Task and Contact related files. Unit testing is primarily used to focus on testing the functionality of individual code components. While I am sure JUnit is not the only possible way to perform this type of testing, it is what was used in each of my test cases. Unit testing is used primarily to isolate application components, through providing input, and then checking that the output given is the output you expected. This can be done in several ways, but I found the most comprehensive and easiest way to do this was with use of the Assert library.

The second testing type I relied heavily on was acceptance testing. The primary properties and purpose of acceptance testing includes testing against business requirements, customer-based testing, black-box testing, user acceptance testing, final stage testing, and defect management. In summation of what all these properties are meant to do, they are meant to test the product against the constraints, requirements, and requests made for the functionality of the program/product, such as password limitation/restrictions for input, or a specific memory amount that is needed to be set aside for a specified purpose, and so on. I used this type of testing quite consistently throughout every method created, to ensure that requirements and specifications from the business (rubric), were met to their expectations, while also keeping quality in mind versus only functionality.

A few testing techniques that I have yet to use are performance and security testing. I would like to learn more about each of these testing techniques in a later course. Performance testing is meant to determine the programs speed, stability, and responsiveness to the system. The main purpose of performance testing is to highlight any areas in the program code that can be altered to make the programs performance better. Security testing is used to ensure the system remains healthy. It is able to do this by limiting the access to approved users only, while also providing protection from hacking, and other security risks that any technology is vulnerable to.

Practical uses for unit testing include identifying coding errors, so that mistakes can be discovered during the early stages of development. It is also intended to ensure the code is modular and easy to integrate with other modules, while also making regression errors as minimal as possible, and maintaining code quality. The implications of using unit testing are fixing the defects which also reduces cost, reduces the debug time, that ultimately saves the developer’s time, and it improves the overall coverage and reliability of the product/software.

Practical uses for acceptance testing are to validate that the software functionality is aligned with user requirements/restrictions, which will also ensure business and customer needs and expectations are met and/or exceeded. Another practical use for this type of testing is to evaluate the software from the end-user’s perspective. The implications for acceptance testing is that it can ensure customer satisfaction by fulfilling their expectations, while also decreasing software production time and providing clients with an effective, efficient, fully functioning, and high-quality end product. Something that may sometimes get overlooked as a benefit of acceptance testing, is its ability to provide a more accurate verification of the systems behavior, due to it running an environment that is extremely similar to actual user’s environment.

The practical uses for security testing are vast but include being used to identify vulnerabilities or other security vulnerabilities that may exist within the software application. It can also be used to ensure the software meets security requirements and standards, and to protect sensitive data or information stored within the program. The implications of security testing is its ability to assist in identifying vulnerabilities while also strengthening the applications security. It can also be very helpful in reducing the risk of security breaches and cyber-attacks, that can ultimately safeguard the confidentiality and integrity of the data.

Finally, the practical uses of performance testing is to test the systems performance under different workloads, measure response time, speed, and stability. It also has practical uses that relate to system optimization and giving it the ability to properly handle high user traffic. The implications of performance testing is that can identify performance bottlenecks before the software is released, increase the systems efficiency, while maximizing its capacity, and also improve overall user satisfaction and experience by delivering a reliable, scalable, and efficient product.

In conclusion regarding test practices, each one mentioned above is an essential part of maintaining software quality and the ability to deliver the highest possible quality products. Each helps to improve the overall quality of the programs code, reduce costs, increase efficiency, identify security vulnerabilities, and meet or exceed the customers’ expectations. I believe the ability to take each project apart and tailor testing methods to best fit the project at hand is the best and most efficient way to approach any new project to ensure that the clients specific needs and requirements are met.

**Reflection**

**Part II – Mindset**

Due to inexperience, for the most part, I adopted a mostly cautious mindset, wanting to ensure all code was tested as accurately as possible while also making certain to provide the most efficient testing coverage. I think I did well at understanding the importance of the complexity and interrelationships of each code file to be tested in an effort to help identify any interactions between code blocks. A good example of how I employed caution is the Appointment files. I did this by designing and the testing the algorithm to handle the complexity of incorrect input, while ensuring the output for those scenarios were correct.

In order to limit bias in my code review, I was continuously reviewing the code to ensure that all test cases passed while executing correctly. Bias is typically a significant concern when we are responsible for testing our own code, due to the likeliness of overlooking any error/faultiness in the code because of my familiarity. The biggest challenge this presents is the chance a developer could subconsciously cut corners and/or overlook possible errors and scenarios.

I do not think I can express enough, the true importance of self-discipline regarding commitment to quality at any stage or role played in development of a program. For software engineers, they need to ensure that commitment by always completing their code with the expectation of producing something of the highest quality. People cutting corners during code writing, modeling, planning, and testing can result in technical debt, which usually leads to future problems that can be extremely costly, not only to the bottom line ($$), but also to the company’s reputation. To avoid this, I plan… well… I WILL regularly review my code, while always making sure to take plenty of time to PROPERLY design and TEST my code, regardless of project size and time constraints. Due to my tendency to overcomplicate and overdo everything, I do see myself having any issue with this in the future. Between my previous statement and a bit of OCD, lol, I think 99% of the time, I will always have ***MORE*** test cases than the latter, to ensure my code is produced of the highest quality I am capable of creating. I think have shown this to be true in every assignment we have had throughout this course, by always trying to strive for perfection with my coverage, even to the point of going down the rabbit hole from time to time, not wanting to continue until I saw the perfect coverage of 100% show on my screen, lol.