**7-2 Project Two: Summary and Reflections Report**

Winnie Kwong

Southern New Hampshire University

CS-320 Software Test Automation & QA

Professor Toledo

October 15th, 2023

## **Summary**

* 1. Describe your unit testing approach for each of the three features.
     1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

The best practices I used to approach an alignment with the software requirements was to establish goals and objectives, monitoring the project’s progression, and documentation. Establishing goals and objectives are essential because it helps to determine what type of outcomes should be achieved from testing. I noticed that each feature had similar objectives, so I strategized to prioritize the common variables that needed to be a certain length, could not be null, and creating a variable that must be unique. By completing these objectives first, a huge portion of the project would be considered done because only the contact and appointment had an additional task that needed to be implemented. I also established an effective timeline for the scope of the project. I knew the biggest task was to transition the array into a hash map and estimated that half of my timeline would be utilized to the map and requirements. The remainder of the timeline was focused on cleaning the codes and reviewing all criteria that were met before submitted the final application. Throughout the project, I also focused on documenting all experiments. By keeping a record of what was tested, I was able to efficiently ensure I would not duplicate testing and waste time when completing the application. By establishing goals and objectives, monitoring the project’s progression, and documentation, I was able to be focused my alignment to the software requirements.

* + 1. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

JUnit testing allows developers to create a reliable, efficient, bug-free infrastructure. The quality of the JUnit test was a success, with 100% coverage for each class within the project by focusing on creating a checklist for the testing activities, identifying gaps, and removing any redundant code. When creating the test files for the application, I made a checklist of all the tasks that need to be implemented and tested. By having a list, I could reduce the risk of errors while aligning closely with the functions. The next step was to focus on all the gaps from the missing branches that were not covered. By shifting my focus to the missing branches, I could identify the areas within my application that were not covered by the current test cases and expand the scope of the coverage. According to Shah, “Removing cloned code can improve test coverage ratios in the same way as removing dead code.” (Shah, 2021, para 64). If the application had been submitted the same as the milestone assignments, the code would have been just as bulky and complex for unnecessary reasons. Removing redundant codes increased the readability and enhanced the code’s quality when using JUnit. By looking at the coverage, we can confirm the test quality has excellent performance due to the lower chances of identifying bugs within the application.

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

My code is technically sound by following industry-standard practices such as appropriate syntax for Java, following Java naming conventions for readability, and proper comments to improve maintainability. Having the right syntax in Java means every line of code runs within its respective class. Each variable is appropriately declared and initialized; for example, in ContactService.java, lines 14 to 89 illustrate accordingly to ensure the code compiles and runs successfully. Java naming conventions enhance readability because if developers need to fix a code, they will spend less time trying to locate the correct method. When looking at the constructor in Appointment.java from lines 20 to 40, each variable is defined with precisely what is required within the file, such as appointmentID, appointmentDate, and appointmentDescription, making it clear that each appointment only has an ID, date, and description. According to Vartanian, “Code commenting helps make codebases and projects more maintainable.” (Vartanian, 2022, para 11). Proper comments in a code are vital because they provide clarity for other developers during collaborations. An example is under TaskService.java from lines 31 to 39. The statement explains it is a method that deletes a task only if the task exists, or else it will throw an illegal argument. By following industry standard practices with appropriate Java syntax, naming conventions, and commenting, I was able to create a code that is uniform, maintainable, and easy to read without using excessive lines.

* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

My code is efficient, using an effective data structure, extracting methods, and using the final keyword. I enhanced the performance by using an effective data structure such as a hash map because it allows faster lookup. Within the AppointmentService.java file, line 23 to 41 executes a function to add and delete an appointment. Using a hash map, I can efficiently grow and shrink the map as needed without copying the data structure, making the performance smoother. Extracting methods help to reduce any repetition, making it easier to read code. In the file Task.java, lines 36 to 42 show a Boolean for validation inputs. The validation input helps to consolidate any redundant logic into one method, making it easier to reuse in the future. Lastly, having a final keyword can help reduce errors in a code. (*12 Tips to Optimize Java Code Performance*, n.d.). In the file Contact.java on line 12, the contact ID uses a final keyword to ensure the code cannot be changed, meeting one of the project's requirements. By having an effective data structure, extracting methods, and using the final keyword, the code is efficient and ready for deployment.

# **Reflection**

* 1. Testing Techniques
     1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

Some of the software techniques employed in the project were unit, acceptance, and integration testing. Unit testing breaks the application’s code into small units to confirm if the unit works as intended. A successful unit test is when a unit can execute quickly while locating defects within the unit. Acceptance testing assesses if the application meets expectations before deployment. Acceptance testing focuses on the entire system while being efficient when finding bugs that interfere with the application’s performance. Lastly, integration testing ensures that the code works together efficiently. With intergradation testing, all components in the application are validated to achieve the desired functionality while detecting system-level issues. Using unit, acceptance, and integration testing techniques, the application is at low risk for bugs and, most importantly, has a reduced risk of failure.

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details. **<Write your answer>**

There are many different techniques to use for software testing. However, some that could have been more suitable for this project are performance, maintenance, and usability testing. Performance testing checks “your website and app performance, which may extend to testing servers, databases, networks, etc.” (Cohen, 2022, para 11). Performance testing because the objective of the project does not include network testing and does not have a website or app. Maintenance testing focuses on the quality of the application after deployment and changes are made. Maintenance testing would not apply to this application because the project is completed but unreleased to the public. Lastly, usability testing requires participants to evaluate if an application is easy to use and understand. Usability testing does not apply to the product because the project lacks an interface. Although performance, maintenance, and usability testing are great techniques, it does not apply to the same context as the project.

* + 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations. **<Write your answer>**

Unit, acceptance, integration, performance, maintenance, and usability testing have unique practical uses and implications affecting various applications. A calculator could use unit testing to determine if the outputs are correct despite the different input values. Acceptance testing could be used to collect data from participants trying out a prototype of a new product, such as a smart washing machine. Participants would provide feedback to determine if the product is viable. An example of integration testing would be seeing the effects of adding a new comments section to an existing website. Developers must confirm how the comments section would work with the current system. Since performance testing relates to speed, stability, and scalability, it could be used to test a video game. The performance testing can determine how many users could be on a server before the game crashes. Imagine a bank implementing an update to send paperless statements. Maintenance testing would apply to this scenario to ensure that all preexisting components would still function properly. A scenario to use usability testing is performing an interface assessment for a phone application. Users could soft test the product to determine any last adjustments before release. Unit, acceptance, integration, performance, maintenance, and usability testing are techniques that apply to many real-world scenarios.

* 1. Mindset
     1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

Throughout the project, I was very cautious and took time to identify all the requirements thoroughly. Appreciating the complexity and interrelationships of the code being tested is essential because each class has its unique feature and can help determine if the application is a success or a failure. An example would be using unit testing to update the contact's first name. By updating the first name, the function needs an accessor and mutators to change the first name. When testing if the process is successful, the application focused on the outputs that are inside and outside the parameter to ensure the correct results were returned. If the developer can understand how the code works, it is easier to create an application without breaking other portions within the system.

* + 1. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

I limited the bias in my review code because it leads to false assumptions that are misleading and harmful to the code. An example would be the requirements of the variables that have specified requirements within the application. When testing, if the developer only focused on what is needed for the conditions, the developer would only validate the expected behavior rather than including the negative validations. Negative validations play an essential role in coding because they also ensure the system behavior behaves appropriately and that invalid inputs are handled accordingly. (*Negative Testing*, n.d.). Using both positive and negative validations allows the code to be more stable and reliable. If a developer is not open-minded, it may interfere with the ability to understand how the application works.

* + 1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

Discipline is essential when committing to quality as a software engineering professional because it allows the developer to create efficient and maintainable code. Cutting corners can cause issues such as being prone to bugs that can be more difficult and expensive to fix in the future. In coding, technical debt is the aftermath of a poorly written code that becomes a priority to resolve. (*Technical Debt*, 2021). Developers can execute different practices to avoid technical debt, such as writing clean code or refactoring. Another way to avoid technical debt is to test the code. For example, within the project, JUnit testing was used to catch bugs and issues before the application was deployed. It takes a lot of time and practice to be disciplined and committed to the quality of developers, but it's worth the wait to illustrate performance and efficiency.

**References**

*12 Tips to Optimize Java Code Performance*. (n.d.). <https://www.tutorialspoint.com/12-tips-to-optimize-java-code-performance>

Cohen, N. (2022). What is Performance Testing? What is Load Testing? What is Stress Testing? A Comparison. *BlazeMeter*. <https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing#perf>

*Negative testing*. (n.d.). smartbear.com. <https://smartbear.com/learn/automated-testing/negative-testing/#:~:text=As%20you%20can%20see%2C%20negative,application%20more%20stable%20and%20reliable>.

Shah, H. (2021, June 28). A detailed guide on test coverage. *Simform - Product Engineering Company*. <https://www.simform.com/blog/test-coverage/>

*Technical debt*. (2021, September 2). https://www.productplan.com/glossary/technical-debt/

Vartanian, E (2022, February 14). *6 coding best practices for beginner programmers*. Educative. <https://www.educative.io/blog/coding-best-practices#comments>