**Project Two Journal**

Yaumel Betancourt

Southern New Hampshire University

CS 320

Prof. Blair

August 24, 2025

The mobile application required extensive testing of its three services i.e., appointment, contact, and task services, to ensure that each facet was working as intended. My approach was fully aligned with software requirements because all the required functionality is present in each service. In the case of the task and contact services, my tests made sure the character lengths were correct for all fields and did not exceed the max lengths. In addition, for the appointment service, I made sure to test that my implementation did not allow to set a date that was past the current date. My JUnit tests were thorough making sure that I covered the different possible cases from invalid lengths to null values. My tests were effective based on 100% code coverage making sure each line of code was tested at least once. The code I wrote is also technically sound since I applied many best practices to ensure readability. For example, in the task class I avoided magic numbers by creating constants for the max lengths such as MAX\_LENGTH\_10 so anyone that reads it understands what the numbers represent, not to mention it can be reused such as for the name. I also made sure to add documentation to the methods to further explain functionality, purpose, and restrictions as per requirements. I made all the class members private and used getters and setters to protect the state of objects via encapsulation. I made sure every method had a single responsibility, i.e., get, set, update, etc. The major way I improved efficiency was to use a hash map when searching in the service classes for each of the services. For example, in the contacts service class, I used the id as a key that would point to a contact object. This is more efficient, particularly with larger data sets, than having to go through an array, one element at a time, trying to find a match. This approach was used when adding, updating, or removing an entry.

Since I wrote the code, I used white box testing to write several unit tests for each service. The first technique I used was boundary value analysis. This made up the bulk of my tests since most of the requirements were concerned with max character lengths. My tests included inputs with valid character lengths and character lengths that exceeded the max length by one character. I also tested for null values to make sure my code handled those cases correctly, in this case with using exception handling. The other technique I used was use case testing. For each service, I would test if the service class was correctly adding, updating, or removing entities correctly. One example is the task service class. I made sure the add method correctly added a new entry, the update method updated an existing entry, and the remove method deleted an existing entry since those are the main ways users will interact with the task service. One crucial technique I used was state transition testing. This was used in each service class for each service when adding, updating, and removing entities. For example, when adding or removing an appointment in the appointment service class, I made sure the count had either increased by one or decreased by one respectively. One method I did not need to use was equivalence partitioning. The conditions in this case were binary in nature, i.e., whether the value was valid or not. The partitioning was not complex enough to merit using equivalence partitioning for testing. I also did not need to use a decision table since the cases were simple enough. All of the techniques I mentioned can all be highly beneficial during the software development process. They provide ways to test boundary conditions which tend to be the cause of many bugs, they help make sure a system is in the correct state and can change state as intended based on business requirements, and they help organize tests by grouping them into subsets based on behavior or related system components.

Testing is always tricky, in particular when you are also the person that wrote the code. This introduces a bias that can be easily overlooked if one is not careful. I made sure to exercise the greatest amount of caution to counter my own bias. First, I made sure that I achieved complete code coverage to ensure that the code in its entirety was working correctly. After that I started to think as someone that did not have access to the code but simply interacted with the application. I asked myself how would someone use this? How would they go about adding an appointment for example? What if they forgot they had set an appointment and tried to set it up again? I essentially tried to figure out if there was a way to break the application and then wrote defensive code to prevent failures, or at the very least, have the application handle failures gracefully. As the developer you may overlook such a scenario because it seems silly. Why would someone try to book the same appointment. Right then and there you fell for a trap because you must code defensively and assume that if there’s a way to break an application or that there is a good possibility the user will not go down the “happy path” as it’s called when the user interacts with the application as intended without deviating from the intended use. It is just as important to write clean, simple, and reusable code. This helps reduce technical debt in the future. Even if you are the only one working on a project, you are only making things harder for yourself when you go back and forget what exactly a method is doing or why a particular algorithm was used. It is impossible to test every scenario and catch every bug, but developers must remain disciplined and methodical to catch as many defects as possible. In this case, the app is innocuous, and errors do not result in catastrophic failures, but the same cannot be said for flight software or software used in medical equipment. I believe there is an ethical obligation to develop the best quality code and to test as best as possible.

**References**

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019, July 26). *Software testing: An ISTQB-BCS certified tester foundation guide* (4th ed.). BCS Learning & Development Limited.