Shimin Chan

CS 320

Prof. Bermudez

**Summary and Reflection Report on Project One**

**1. Summary**

During the development of this mobile application, I initially used an informal review technique and dynamic testing, such as specification-based testing approach, to ensure alignment with the software requirements. I began with informal review techniques, such as desk check and dry runs. I explored and navigated the application to identify any vulnerabilities. After that, I examined the code line by line to ensure that the application functions correctly, while also checking for any typos or duplicate tests. Moreover, I wrote unit tests based on the application’s functionalities. For instance, in the Contact Service module, the application is designed to create, update, and delete contacts. I created unit tests for adding a contact with all valid inputs, as well as for updating the first name, last name, phone number, and address. I also test all these fields with “null” to ensure it throws an error message. Given that the requirements included a unique and immutable contact ID, I also tested updating the contact ID and creating a duplicate contact ID to ensure that both scenarios would trigger the appropriate exception handling. Finally, I tested the deletion of both existing and non-existing contact IDs to confirm that an existing contact ID is successfully removed from the list, while attempting to delete a non-existing contact ID properly raises an exception without breaking the application. Additionally, the requirements impose constraints on all input values. For example, in the Task Service module, the task name can only be 10 characters or fewer. I included tests for inputting 9-character, 10-character, and 11-character task names. This ensured that both 9-character and 10-character task names could be successfully created, while an 11-character name would raise an exception to handle the error. Overall, the quality of the JUnit tests was quite effective, achieving a coverage percentage of 82.6%. However, the more important aspect is that these tests address the most critical elements of the product requirements, such as input validation, ensuring immutability for all ID fields, and guaranteeing that all functions operate correctly.

To ensure that the functions are working correctly, I implement edge cases and negative testing. For example, in the Appointment Service module, valid appointment dates are any days that occur after the current day. I set two sets of dates in the future and one date in the past.

A screen shot of a computer program

AI-generated content may be incorrect.

When using the two future dates, the application can successfully update the appointment.

A screen shot of a computer code

AI-generated content may be incorrect.

Conversely, if the past date is used, the application will display an error message indicating that it is unable to create or update the appointment.

A computer code on a black background

AI-generated content may be incorrect.

To ensure efficient code, all modules utilize a Map to store lists of appointments, tasks, and contacts. A HashMap is then employed to retrieve specific objects using their IDs. A HashMap is the fastest way to find an index, allowing it to quickly locate the corresponding page where the detailed information about an appointment, task, or contact can be found. This direct access is achieved through hashing, which significantly enhances the application’s performance and efficiency.  
In the screenshot below, the application deletes the appointment through the appointment ID:   
A computer code with colorful text

AI-generated content may be incorrect.

**2. Reflection**

In the project, my testing approach begins with a more simplistic method check the function's operation and whether the requirements are met, followed by testing edge cases. First, I conducted dry runs and a desk check for an informal review of the code. I went through the code line by line, verifying it against the requirements to ensure everything was correct. Both of these techniques are forms of static testing that help identify errors in the initial stages. They assist in detecting logical or syntax errors that dynamic testing may not uncover easily. The desk check is particularly important in the development cycle, as it "primarily serving as a method to verify and validate the accuracy, logic, and functionality of a computer program or algorithm” (DevX 2014). The earlier we identify an error, the lower the cost of fixing the project will be.

After that, I implemented Experience-Based Testing methods, including Ad Hoc Testing and Exploratory Testing. Experience-based testing techniques are beneficial for situations with unclear or incomplete requirements. Testers rely on their own experiences to target areas they believe carry higher risks in the project. However, this approach can introduce bias during test case development, and if the testers lack experience, it may negatively impact the quality of testing. Despite these disadvantages, this testing approach is still encouraged. "Even when specifications are available, it is worth supplementing the structured tests with some that you know by experience have found defects in other similar systems” (Hambling, Morgan, Samaroo, Thompson, & Williams, 2019).

). Ad Hoc Testing allows testers to explore the software freely without a formal test plan. In contrast, Exploratory Testing is a bit more structured, often involving a mini testing plan or specific requirements and guidelines for testing. Nevertheless, neither approach typically includes formal documentation of a test plan.

Lastly, I used Boundary Value Analysis, a type of Specification-Based Testing, to evaluate edge cases related to the input constraints. Given that this project has clear documentation regarding the requirements, a specification-based testing approach is ideal. This method relies heavily on formal documentation to create test cases based on the specified requirements, which are then executed to determine whether the software passes or fails. Boundary Value Analysis specifically tests values at the edges of valid input ranges. For example, if a number cannot be greater than 10, the test cases would include 9, 10, and 11. In this scenario, this approach provides a shortcut for us, rather than testing the entire set of data. However, this approach might not be suitable for complex projects or when there is a discrete set of input values (Rana, 2023).

There are several testing techniques are not apply to this project, such as Decision Table Testing and Integration Testing. Decision Table Testing is a black-box testing approach where the tester creates a table that lists all possible combinations of inputs along with the expected results for each scenario. This method is particularly effective for complex applications, as it ensures comprehensive coverage and verifies that the software behaves correctly under different conditions. Since this project is relatively simple, we determined that it was unnecessary to apply this approach. Integration Testing, on the other hand, aims to ensure that individual components of a system work together as a cohesive group. It verifies the interactions between these components to confirm they can function together before the software goes into production. We have not implemented this testing yet because the three separate services do not currently interact with each other. As a result, using this test would not be beneficial at this stage.

I believe that testing is a crucial step in the software development life cycle, as it ensures the quality of the product. According to the article "When Coding Goes Wrong", when developers are not "cautious enough" or do not test adequately, it can lead to loss of lives, the spread of false information, and even harm to a company's reputation (Coder Academy, 2016). These consequences are irrecoverable and beyond monetary compensation. Although this is a practice project, I see it as a great opportunity to start taking testing seriously. Once the software goes live, any vulnerabilities can profoundly impact the public. It’s better to take various testing approaches and be thorough in testing the high-risk areas of the project to prevent issues from arising. To minimize bias, I believe that the perspectives of end users, QA testers, and the requirements documentation are key. If I have to test my own code, I will carefully review the requirements document and ensure that the product meets all the criteria on the checklist. As a professional software developer, it's essential to maintain the quality of the product, and testing is vital in achieving that. Cutting corners may save time in the short term, but it often leads to bigger problems later on. Bugs, unstable features, and security vulnerabilities frequently trace back to inadequate testing, resulting in the need to spend even more time fixing them. In my previous experience at a coding boot camp, we skipped writing tests for a feature that later broke during a demo. This experience taught me how fragile software can be without a solid foundation. To avoid technical debt, I will keep practicing on writing clean, maintainable code and test-driven development whenever possible. Catching errors in the early stages is crucial, otherwise, they can lead to larger issues in the future development. Ultimately, I believe that staying disciplined is about respecting the craft, the users, and the future success of the software.

**Reference**

Coder Academy. (2016, May 23). *When Coding Goes Wrong*. Medium

https://medium.com/@coderacademy/when-coding-goes-wrong-e46d84c6565f

DevX. (2014, August 12). *Desk Check*. DevX. https://www.devx.com/terms/desk-check/

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., Williams, P. (2019).

Software Testing: An ISTQB-BCS Certified Tester Foundation Guide (4th ed.).

BCS Learning & Development Limited.

Rana, K. (2023, July 7). *Boundary Value Analysis*. AstOfTesting.

https://artoftesting.com/boundary-value-analysis