Charles Campbell

Module 7-2 Project Two

8 February 2025

Software Testing Summary & Reflection Report

Testing for the mobile application had to align with the software requirements, to secure a smooth operation of the customers’ desired results. As Garcia states, “we need to take special care in the requirements, which guide the whole process, including, of course, the tests” (2017). In regards to the variables in the application, they required an immutable ID and certain variables that couldn’t be null or exceed a certain character limit. With the power of JUnit tests, I could align both the code and the tests to verify that the customers’ requirements were implemented. No ID could be edited, names of contacts and tasks had to be under 10 characters and couldn’t be null, otherwise exceptions would be thrown. JUnit methods of checking for True & False made it not only easy to verify each variable’s requirements and was easily addressable to all the variables in all components of the web application.

The coverage of tests is also essential, as it helps give a non-subjective or ambiguous. It provides detailed measures of info for how the tests are performing. As Hambling and others detail, testing coverage helps both the client to see what has been achieved through a quantitative measurement, while also providing a way of estimating how much more testing needs to be done (2019). Through my own coverage of the JUnit tests, it shows that my test cases covered 100% decision coverage on specific functions/processes, and above 80% statement coverage for the overall program. These detailed reports show, in fact, that both the code I created and the specific tests I ran each individual part and the overall system through were effective; and would show clients that their specific software requirements are implemented, and working as intended.

A computer screen with text

Description automatically generated Keeping code technically sound, or a program that is designed using the best practices and standards, is important because it improves the readability, adaptability, and code maintainability both by the original developer and by potential collaborators. Making sure I put comments on both categorizing and defining functions does play a part in this, but other smaller details greatly enhance the code to be technically sound. For example, the setter functions of the ID variable in all of the classes are set to private. Not only does this follow an important coding standard of encapsulation, but it also covers the customer’s requirement on immutable ID’s for the components. Sound code not only helps the code run smoothly, but it also takes into consideration that system requirements are met.

A screen shot of a computer program

Description automatically generatedEfficient code is also another important factor to consider when designing a system, because it helps ensure that the software performance is optimized, and execution of functions is quick for a high positive user experience. In my own code, I ensured that both the ContactService & TaskService classes utilized a Hash Map, rather than a data structure like a list. This makes a major feature that the customer required, the deletion or editing of a contact, to be (in Big-O notation) O(1). The biggest requirements of removing a contact or changing certain values like a task name or a contact’s phone number becomes nearly instantaneous thanks to the use of an effective Hash Map data structure and it’s functions in Java. This example shows that coding efficiently can greatly help overall code performance, while also achieving requirements in a simple and effective manner. In both the system code and the testing JUnit code, efficient and sound code keeps the system clean, running quick and smooth, and allows for easy future changes and further effective tests.

Testing techniques are essential in any SDLC situation, they allow developers/testers to verify that the system and it’s components are working as intended. Without the use of many testing techniques, code would have sporadic results with costly post-deployment maintenance. The use of both black-box and white-box testing was essential in helping me develop the application. As Hambling & others describe: black-box testing ensures specified behavior in a system is efficient through test cases like boundary value analysis or equivalence partitioning; while white-box testing certifies each component, sub-system and the overall system is correctly exercised (2019). For black-box testing a major example was ensuring that each class’s variables had valid values (like only 30 characters for a description) per the clients requirements; which is practical for other programs that use variables or data manipulation functions with specific conditions. Properly testing in each of the “service” classes that the hash table both worked and would properly edit the objects within them correlates to my use of white-box testing.

A major testing technique I did not utilize was experience-based. This type of testing comes more from the overall background of the developer/tester. “They use the users’ & testers’ experience to determine the most important areas of a system and exercise these areas in ways that are both consistent and likely to be the sites of errors” (Hambling et. al, 2019). With the more specific & detailed specifications of the project, the use of experience testing was not as prevalent as the other two for me. Additionally, the implementation of the requirements in the system was new to me and required no prior experience; so I did not have much opportunity to exercise this type of testing in my development of the application. However, through my processes and thorough experience with this project, I can now take what I’ve learned and done to then apply to future projects and development experiences, in which I can then apply this testing technique more thoroughly.

It's important, as a tester or developer, to exercise caution. This applies from the start to finish in the SDLC. As Hambling explains, the incorrect implementation or testing of software can harm people, companies, the environment and the users if one is not cautious enough to identify errors and correct them immediately; not being cautious can lead to costly consequences (2019). For example, in the code I implemented, if I wasn’t cautious and rushed ahead without testing the data structures, and found errors near the end of development, the entire system would have been unable to function, costing both time and additional resources to fix it.

Bias when testing code, especially when the developer is testing their own work, might lead to errors and some tests getting glossed over. Often this can be attributed to a ‘confirmation-bias’ of having difficulty in the confirmation of finding ones own faults. “If a defect is found in software . . . or in one of the documents on which the code is based . . . the author may see this as criticism” (Hambling et al., 2019). It’s important as a developer to understand that mistakes can happen, and the fixing of these defects is not an attack, but a way to improve the overall system. I made sure to incorporate that, for example, when I modified the data structure from a list to a hash table; my tests showed the list I implemented was not working properly and upon switching to the hash table the system’s efficiency skyrocketed.

Staying disciplined while both coding and testing ensures that no errors get missed. It establishes a foundation of a professional expectation both for the product and the process of the SDLC. Especially in testing, a tester/developer must cover all the bases to ensure a diligent and complete assurance of a fully functioning product. “The discipline of software testing encompasses both static & dynamic testing” (Hambling et al., 2019). From the planning stages, all the way to post-deployment maintenance, a disciplined coder/tester strives to test, so bugs/defects are found and the quality of the product increases to exceed software requirements.

References

Boni Garcia. (2017). *Mastering Software Testing with JUnit 5: Comprehensive Guide to Develop High Quality Java Applications*. Packt Publishing.

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