**Summary and Reflections Report**

**CS320**

**Southern New Hampshire Univ.**

**Stuart Williams**

**February 20, 2022**

**Summary**

I gradually tested my code while in the process of developing. When I felt certain that a piece of functionality would work as intended and ready to test, I wrote a test script for it in its respective JUnit testing class. Following the development practice of testing code modularly and during development allowed me to have confidence that the final product would work as intended and that I was on the right track. This method also ensured that I was developing my code around the software requirements instead of making the requirements fit my development. I feel that my tests are sufficient in satisfying that the desired functionality and use cases are met.

The test coverage was nearly 90%, so I feel even more confident that the code will work as intended and the majority of possible bugs are being prevented. It’s difficult to get 100% coverage, and anything over 80% is satisfactory, let alone 90%. I test every method with conditional checks individually to ensure that the conditions on them are accurately defined. For instance in the Task.java class, each variable has a certain length that it cannot exceed. In TaskTest.java, each variable has its own length check test as well as a null check test. In order to avoid having redundant code, I exported all validation to their own methods. These methods are called whenever a value needs to be set instead of having new code every time that validation needs to be performed. This allows for easier reuse and scalability should the customer wish to expand.

**Reflection**

For this assignment, I was fortunate that the customer provided all functional and nonfunctional requirements prior to the start of development. This allowed for a more efficient process for design, development and testing. For this assignment, I used both static and dynamic testing strategies. Static testing is a software testing methodology where the code is not executed when it is tested.

Static testing is used to find bugs in early states of development (since the software may not be able to run yet) and make sure that the code meets all of the required functional criteria. There are many different practices of static testing, including: software inspection, structured walkthroughs and technical reviews; although, I only used software inspection for this assignment. For larger projects with more intricate pieces, technical reviews with peers would be very helpful as fresh sets of eyes are more likely to find hidden bugs.

Dynamic testing is a software testing methodology that tests how the code functions and its behavior by executing the code. Dynamic testing is used to make sure that the code functions as intended with various inputs and parameters. Different practices of dynamic testing include white and black box testing, but I only used white box testing for this assignment. White box testing is testing when the “internal structure/design” is known to the tester and black box testing is when the structure/design is not known (Hamilton, 2021). Had this project been on a larger scale, it would be important for black box testing to take place, as it’s likely mistakes are avoided when the tester knows the inner workings of the system.

When I test my code, the first thing that I try to do is ensure that it functions as intended. Once I have my code in a working manner, I then adopt the mindset of trying to break it. I see what inputs it can handle, how it behaves whenever it gets something that it shouldn’t, and what happens. This ensures that the code is robust enough to handle user and operator error and won’t crash on them if they fat finger a semicolon in the name input field. It is important to find all the bugs possible during the testing phase to avoid a 3 a.m. phone call from a customer because their system crashed.

Bias when testing your own code is hard to avoid. Without realizing it, you will avoid doing things that are unexpected and could potentially break your code because you know how it works and what it wants. This is why it is important to either have other people test your code or to do your best to try and break your code during testing. This is why I try to follow the two step process I outlined in the previous section: get your code to work, and then try to break it.

I do not like to put my name on code and ship it if I am not confident that it will function as intended. It’s my goal as a developer to write solid code that works and to have a reputation as a good developer because it’s almost guaranteed that in the future someone else will end up working on code that I developed, and I don’t want them to groan when they read my name in the header. Furthermore, I have had to go back and look into projects that I completed years prior to expand on. It is much easier on myself and others if my code is clean, well-documented and follows best practices. Cutting corners may save time now, but 6 months after the project ships and the customer calls you at 3 am because their production line is down, you’re going to wish that you spend an extra day or two to test.

References

Hamilton, T. (2021, December 25). What is Dynamic Testing? Types, Techniques & Example. <https://www.guru99.com/dynamic-testing.html>