**Project Two: Summary and Reflections Report**

Connie Knupp

Department of Computer Science, Southern New Hampshire University

CS-320 - Software Test Automations and QA

Professor Kalysa Wilson

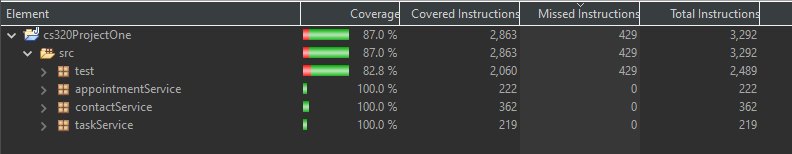
April 21, 2024

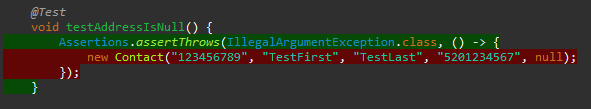
**Summary**

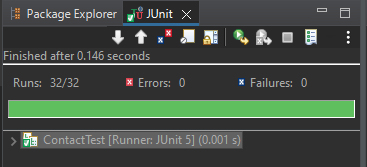
Writing both the software and the tests was closely aligned to the software requirements presented in each module. I began with writing code that met the requirements and would be easy to modify if the required lengths or limits changed in the future.To write the tests for each of the three features I started with a test to see that the feature worked as intended with inputs that fall withing the software requirements. I then created tests with inputs that were specifically created to fail the software requirements. I attempted to make a test for each input that should be not accepted per the software requirements, in this way, I could ensure that the software only allowed inputs that met the desired requirements. This is visualized in the testing example below, captured from my AppointmentTest class. These tests were made to test the software requirements that stated “the appointment object shall have a required unique appointment ID String that cannot be longer than 10 characters. The appointment ID shall not be null and shall not be updatable.” (SNHU, n.d.).



This method provided effective coverage as when the test module was run as coverage for the source code each class had 100% testing coverage. This does not mean no further testing is needed, but rather that each branch of logic in the software below was covered by one of the tests. Further testing would cover integration with the user interface once developed, the interface testing should ensure all inputs are wrapped to safeguard against code injection. Additional testing would also cover things such as speed and scalability which cannot be tested as thoroughly at this stage of development. The JUnit tests are used to ensure all branches of logic work as intended before we have made it to a fully functioning system, testing at the earliest stage possible reduces technical debt or cost on the project. The screenshot below proves that the JUnit tests provide 100% coverage on all three services as coded thus far.



Writing the code for the services requested as well as the JUnit test made it easy to adjust the code early in the process to get the expected results. There were several times that I copied code from one service to the next and used the find all and replace all functions to change the needed variables. This meant I just had to change lengths to meet specifications and made it far more efficient in writing the code and tests. For example, I could copy the testContactIDTooLong function and change “Contact” to “Task” to use the same function for testTaskIDTooLong. I could tell my code as technically sound when all JUnit tests ran without errors or failures. This indicates the tests threw exceptions when expected or worked without exception when expected. The code below was expected to throw an exception, the requirements state the address cannot be null, the red indicates an exception as thrown as expected.  Similarly, an indication the code is efficient can be inferred from the time it took to run the tests, the Contact class had the highest number of input fields to test, meaning the ContactTest class had the most tests contained. Even though it was most expansive running the JUnit tests took less than a second as indicated in the screenshot below.



**Reflection**

**Testing Techniques**

The testing techniques I employed during the project were dynamic white box decision testing. White box because it requires knowledge of the internal structures and decisions in order to test their functionality. The tests I employed were dynamic because they involved executing actual code. “Decision testing exercises the decisions in the code. In a similar way to statement testing, tests are designed to attempt to force the program to execute particular decisions in particular ways.” (Hambling, et al. 2024). The JUnit test are therefore decision tests as they are designed to execute the decision points and cover all branches in the logic. The testing techniques not used in this project were black box testing which does not require knowledge of the code or internal structures. Black box testing includes static testing, used to test whether work products such as requirements documents meet the client’s goal for the program. Black box testing can also include tools like decision table testing in that outlines what conditions lead to certain outcome, although these can be very useful depending on the project they were not used in this project as they would not be the best fit for the logic in this project. I also did not include integration testing or use case tests as the entire system is not yet available to test the full utility of the project.

**Mindset**

My mindset when working on a project is to follow the methodical strategies. I use a systematic approach to develop tests that cover all logic branches and ensure the product meets the software requirements. This worked well with the stage of development that this project was in. Though to be fully cautious additional testing would be required as the project continues to ensure the classes developed integrate with the user interface appropriately and to ensure the full product meets user’s expectations and requirements. The complexity and interrelationships of the code need to be considered as the code is developed to ensure the testing covers both successful branches of logic as well as fails when expected. Testing that does not cover all aspects would lead to a product released with defects or bugs that can affect performance.

As a software developer I tried to limit bias on review of the code by relying on the coverage tool to quantify the extent of the test coverage. This approach is less subjective and minimizes bias, it gives a neutral look at how well the decisions are covered by testing. I can imagine that as a developer responsible for testing my own code bias would be a large concern especially as more time as been dedicated to the project. Personally, I would prefer to be the one that recognizes a defect and corrects it before it can be found by others. In this way I can motivate myself to be a critical and thorough tester. A good example of this is the first code I wrote for the Contact class had a ton of redundancy. While it met the stated objects, I realized it was not best practice. If they length of ID numbers were needed to be changed in the future, the developer would have to change the code in too many places. I redesigned the code to make it cleaner, more inline with best practice. I prefer seeing areas of improvement before others do so that I can present my best work to those who with me on a project.

Not only does this being methodical and disciplined help you to become associated with higher quality of work but it is important in our field so that we do not release insecure or unsafe products that may have disastrous affects on the lives of our clients. An example of this would be the Horizon software bugs and the wrongful prosecution of the UK Post office branch managers. Their poorly performing software resulted in “more than 900 post office small-branch operators were wrongly prosecuted for false accounting and theft based on Horizon information.” (Bartlett-Imadegawa, 2024). Living with the knowledge that cutting corners or ineffective testing lead on your part contributed to the catastrophic events would be horrific but you would also lose credibility and customers when you become associated with poor quality work. The far-reaching consequences of a release software defect are the biggest reason why it is important to remain dedicated and disciplined when writing code and testing said code as thoroughly as reasonably possible prior to release.

References

Bartlett-Imadegawa, R. (2024, January 19). *Fujitsu’s role in U.K. Post Office scandal: 4 things to know*. Nikkei Asia. https://asia.nikkei.com/Business/Technology/Fujitsu-s-role-in-U.K.-Post-Office-scandal-4-things-to-know

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.

Southern New Hampshire University; SNHU, (n.d.). *Project one guidelines and rubric* [Class handout] Brightspace. <https://learn.snhu.edu/d2l/le/content/1535939/viewContent/30992456/View>