Zachary Hartt

CS 320

Southern New Hampshire University

June 14, 2022

Unit Testing

Prior to this lesson I had no personal experience utilizing JUnit Testing. In each project I had consistently used an exhaustive testing method: brute forcing my knowledge of the program to test the “completed” project as if I was a user trying to utilize and intentionally break the program. This would identify any vulnerabilities and issues that I could think to test, but it did not provide a reliable means of testing; it was not feasible to achieve 100% test coverage under these circumstances.

With the implementation of JUnit Testing in my projects, more specifically in the Contact Service and Task Service projects, I was able to verify full coverage of code being tested and I was able to validate that the code was functioning properly without issue. With just a few extra lines of code I had verified coverage of my Task Service project and validated that the functionalities were working to the fullest extent and to the guidelines of the client’s requirements.

In the Contact Service project I was able to test each requirement by verifying, for example, the unique contact ID was not null and that it did not exceed 10 characters in length (***Figure A)***. This is just one example of how I ensured I followed the guidelines in this project.



***Figure A***

The Contact Service project contained my first utilization of JUnit Testing. It was not until later that I had realized I did not have full coverage of my classes in my test cases. I corrected this in the Task Service project by verifying I created a test for each function and requirement of the project, this would ensure full coverage of my code to create a viable test of the code (***Figure B***).



***Figure B***

Software Testing Techniques

Throughout the milestones I utilized various software testing techniques. I vastly explored my options for testing as to learn multiple different methods of testing my projects. At first I had created an object class, initiated the object, and verified the parameters were stored correctly for that object. Then, I did a similar procedure when I created methods for updating the variables; I verified these variables updated after I invoked these methods.

As I explored more options, I dove into JUnit testing. This form of testing allowed me to test multiple aspects of each area of my code and verify it was working properly based on what information I fed it. I also latter learned about coverage testing, where with my JUnit test I can validate my test case was covering all of the code I was intending to cover.

Due to these projects being relatively small and just myself coding them, there were many testing techniques I did not utilize. Some of these include peer reviews, decision tables, state transition testing, among many other options available to any software developer. The techniques I chose to utilize were basic, full, accurate, and appropriate. They were not advanced techniques, but they covered all of my code base and cases, they provided accurate details into any issues or if there were none found, and I found the techniques appropriately utilized in my projects based on what I was working on.

Mindset

As I was testing the code and outcomes, I ensured I was covering every aspect of the code, including all elements – as to ensure each requirement was being met for the client. I did this to ensure everything worked as expected. Depending on the code length and size, it can become very complex and confusing – thus thorough testing is required to ensure a proper outcome. Testing your own code can be bothersome as you know the code line-by-line, meaning you know how to make it work. If you are testing your own code it is very hard to think as the client – having the clients test the outcome is a much more efficient way of ensuring the proper use of the features you have created.