**Wesley Blackwell**

**CS 320 Project Two**

1. Describe your unit testing approach for each of the three features.
   1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

I used a series of tests to verify that the variables were not equal to null using the “assertEquals”, “assertFalse”, and “assertTrue” functions. Each assignment had similar requirements but each had one small change. I set up my requirements in the class methods to throw an exception in order to utilize testing methods later. I was able to run tests and looks for exceptions. The delete method gave me some trouble until I did some research and found some good examples and explanations.

* 1. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

I struggled quite a bit with my coverage percentages for the assignments. I had difficulty achieving greater than 60% coverage at first. Later, after finding examples I was able to raise my percentage but I did not get to 100%. I had such a hard time with the add, edit, and delete methods at the beginning that I forgot to add the null and length tests. I got fed up with it. The lack of examples and designs for code to testing relationships made it hard to perform. It took me a while but I got better. By the third assignment I had an okay system. I feel confident that if I did a few more assignments of this nature, I would be more fluent and productive.

1. Describe your experience writing the JUnit tests.
   1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

For the unique I.D. requirement I used the random function of UUID to generate a number and restricted the number of characters to the first 10. This allowed me to stay within requirements. In the first two assignments, the constructor used the “**this**” function to set variables. In the third assignment, I changed that up to have the setter methods do it in order to check requirements with less code. This allowed my tests to verify that the setter methods were functioning and checked requirements validity. In the beginning, I intended the main method to generate the unique I.D. In the end, I was using the add method in the service class to generate it and send it to the constructor. To streamline things, I created a find method to search the list for an object and return the I.D. This was helpful in creating the tests for the service methods.

* 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

I combined my variable requirement constraints in one line of decision branching. Another item that helped consolidate my code was the search method I made to the service class. The random function of UUID really helped me create unique strings of digits. Restricting the string of digits was easy with the built in methods of Java. One code example:

For (Appointment appt: appointmentList)

{

If (appt.getAppointmentId().equalsIgnoreCase(id))

{

System.out.println(“ID found”);

return appt.getAppointmentId();

}

}

1. Testing Techniques
   1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

Static testing was the goal of each assignment using Junit tests. Visual inspections, or reviews, were first. Then, I wrote the Junit tests to check that my code was properly working and meet the requirements.

* 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

We did not employ any dynamic tests for any assignment. There were no security test performed such as a Maven dependency check. There are many forms of testing we didn’t use. Regression, integration, acceptance, system, and other forms of testing were not used.

* 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.

Code reviews are useful by catching simple code errors such as missing semicolons or other simple mistypes and errors we humans make. Junit testing was used to test the individual parts of the application. Maven dependency checks search for known exploits and other known security issues. Integration testing checks integral parts of the system for proper communication and function. System testing verifies the function of a system as a whole.

1. Mindset
   1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

As a software developer, I employed caution when writing my application by writing the methods with testing bias. This allowed me to write simpler and more specific tests. One example of the code I used to help myself simplify the relationships and complexity was the search method in the service class. It was important to recognize and appreciate this complexity of the assignment. I created this additional method as a precaution for testing. This method simplified my test coding greatly.

public static String findAppointment(String id) throws Exception

{

For (Appointment appt: appointmentList)

{

If (appt.getAppointmentId().equalsIgnoreCase(id))

{

System.out.println(“ID found”);

return appt.getAppointmentId();

}

}

throw new Exception(“ID not found”);

}

* 1. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

Bias can be a killer for software. My own bias got me because it caused me to overlook mistakes in the code. I missed errors and left of requirement verification in some parts of each assignment. In the first assignment, I overlooked the unique ID verification and generation. In the second assignement, I forgot to check for the null requirement and the unique ID requirements. In the third assignemtn, I forgot to verify the length of the strings for each variable. I was so focused on fixing my mistakes from before that I made a new mistake.

* 1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

Cutting corners can be catastrophic like the spaceship that lost control because the radius variable was incorrect. That is a five hundred million dollar mistake. Simple unit testing would have caught this mistake. Cutting a corner can cause a simple mistype or a capital letter to go unnoticed. Misrepresentation of variables can cause wrong data to be moved around and can be detrimental to the system or other entities. In the case of the spaceship, it was a financial hit and could have been catastrophic if the ship fell into a major city.