**Summary**

**Farrik Barnard**

**CS-320**

1. **Describe your unit testing approach for each of the three features.**

My approached was to thoroughly test my code by reviewing the requirements and test accordingly. When I was coding the task requirements, like “The task object shall have a required name String field that cannot be longer than 20 characters. The name field shall not be null.” To test this requirement I tested that it could be less than 20 characters, equal to 20 characters, and that it would error. Which is what the requirement asked. Then I tested for a null value, which was also part of the requirement, and that also failed as expected. To be honest I hard a hard time understanding how to write the unit tests. I have written unit tests in the past that were done by a tester to test the system. This process was an eye opening experience on how to set up and use testing in a more efficient way.

When it comes to the coverage I was at around 64 percent. I know the ideal percentage is 80 percent. Some of my tests I thought were valid and the tests themselves worked as they should. I knew my tests worked by looking at the results. If a test failed, that was suppose to, like if an input went over 10, 20, or 50 characters. The test would fail because the input would be invalid and not be accepted.

1. **Describe your experience writing the JUnit tests.**

To write technically sound code, the code should be tested thoroughly and often to ensure the code is of good quality. Thorough testing is one of the most crucial things you can do to make sure your code is functioning as intended. Testing can assist you in identifying and resolving flaws, failures, and vulnerabilities in your code before they become issues for your clients or users. For the contact test I had the following code:

// test for first name

@Test

**void** testContactFirstNameIsMoreThanTen() {

**var** newContact = **new** Contact("10001", "Farrik23423423432", "Barnard", "6096657878", "4011 Great Ln. Party, NJ 08754");

*assertTrue*(newContact.getFirstName().equals("Farrik23423423432"));

}

@Test

**void** testContactFirstNameIsLessThanTen() {

**var** newContact = **new** Contact("10001", "Farrik", "Barnard", "6096657878", "4011 Great Ln. Party, NJ 08754");

*assertTrue*(newContact.getFirstName().equals("Farrik"));

}

@Test

**void** testContactFirstNameIsEqualToTen() {

**var** newContact = **new** Contact("10001", "Farrik1234", "Barnard", "6096657878", "4011 Great Ln. Party, NJ 08754");

*assertTrue*(newContact.getFirstName().equals("Farrik1234"));

}

// test for null first name

@Test

**void** testContactFirstNameNull() {

**var** newContact = **new** Contact("1001", **null**, "Barnard", "6096657878", "4011 Great Ln. Party, NJ 08754");

*assertNull*(newContact.getFirstName().equals(**null**));

}

This code tested the first name input. This test was set up to create a new a new contact and insert valid and invalid input to test functionality. It tests if less than 10 characters are accepted, equal to 10 characters were accepted, and if more than 10 characters were accepted. The first to test passed as expected. The more than 10 failed because it was an invalid input. This shows that the code is working as it should. It does not allow more than 10 characters to be entered. The requirement stated that the ID should not be more than 10 characters and the code working as the requirement states.

1. **Reflection**

**Testing Techniques**

Some of the software testing techniques that I employed were automated tests and functional tests. Automated testing is used to replace the labor-intensive process of software product review and validation. These days, automated testing is typically incorporated from the beginning of a software project. This allows greater focus on developing quality code that will meet the requirements of the stakeholders. Functional testing looks to determine if every aspect of the application functions in accordance with the software requirements. To determine if a function's output meets the expectations of the end user, each function is compared to the related requirement. These two approaches work in unison. The automated test will test the functionality of the code so that we know it meets the requirements.

One software technique that was not employed was a human simply testing the code. Running through the requirements and making sure the code is working as expected. This process could be very time consuming and also prone to human error. One day you may think you tested everything, but maybe forgot one step that could cause delays or defects.

These techniques can be used in many situations to help developers code projects faster to meet deadlines. Meet stakeholder expectations and make sure the code is meeting all requirements.

**Mindset**

My mindset was to throw caution to the wind. You have to break a couple eggs to make an omelet. When coding the appointment and appointment service code, the understanding they would be working together was crucial. In the appointment code we have a getAppointmentId(). This was used in the appointment service code to look for the appointment id. Without this the id would not be found and nothing would be updated.

So I believe there will always be bias in the code review as the developer. When working on my last job we always used someone else to test so that the biases would not be a part of the testing process. Once I was coding in Ruby and I had a front end check using JavaScript to make sure the input field would not accept anything but numbers. Me thinking there was no way I could mess this up didn’t really write a good unit test when I passed it off to a tester. Well the tester was through and found that my code failed the requirement. If I were the tester I most likely would not have tested the way I should because I thought I had it working correctly.

As I stated above with my own experiences. Cutting corners is a for sure way to get fired and cost your company money. If you have self-discipline you can effectively manage your time by doing activities in a timely manner, and deliver results on schedule. The work quality will suffer if you hurriedly write code. You can take your time, carefully go over your work, and make sure you're producing a high-quality product when you have self-discipline.