**Brandon Hornick**

**CS 320**

**June 19th, 2025**

**Module Seven Journal**

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

My unit testing approach of the three features in this project, the Contact, Appointment, and Task features, were aligned entirely with the software requirements. In each assignment, we were given a list of requirements for the feature that needed to be met for the program to be satisfactory. Using JUnit testing, each individual requirement was tested referincing the original requirement list. For example, in the Appointment feature, we are given:

* The appointment object shall have a required unique appointment ID string that cannot be longer than 10 characters, shall not be null, and shall not be updatable.
* The appointment object shall have a required appointment Date field.
* The appointment Date field cannot be in the past, shall not be null.
* The appointment object shall have a required description String field that cannot be longer than 50 characters, and shall not be null.

To use testing to ensure these are met, I implemented a test for both a satisfactory case, and a case that shouldn’t be accepted for each requirement. This ensures that the program meets the requirements and fails when the requirements are not met.

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

To ensure that my code was technically sound, I validated all of the requirements and potential issues using JUnit testing. Specifically, I used JUnit’s *Assertions.assertThrows* and *assertTrue*. For example, to test that an invalid contact ID throws an exception, I used:

Assertions.assertThrows(IllegalArgumentException.class, () -> {

new Contact("12345678910", "Brandon", "Hornick",

"3041231234", "123 Central St");

});

This code checks tests that the contact ID constructor correctly checks for the 10 character limit, ensuring the class behaves as required.

To ensure that my code was efficient, I used shared testing variables, and concise, reusable logic. For example, Instead of duplicating the constructor calls in each test, I initialized the contact in a BeforeEach method:

@BeforeEach

void setUp() {

contact = new Contact(id, firstName, lastName, phone, address);

}

Doing this in my code minimizes repetition and ensures consistency across all test cases. It also makes it easy to update the initial values in one place if needed. I also used comments and separated all tests by the specific requirement they were testing. This allows for readability and updatability in the future.

1. **Reflection**
   1. **Testing Techniques**

The software testing techniques that I employed during this project unit testing, boundary testing, acceptance testing, as well as manual testing. The unit tested was done through JUnit, which essentially allows you to create your own test cases with your own input. This means it is up to the developer to create accurate tests, and to hit all the points needed. The JUnit tests are best used in cases where you have conditions that objects need to comply with, such as requirements. The two other types of testing, boundary and acceptance testing coincide with the unit testing. Boundary testing involves checking the boundaries of the given conditions. For example, if we are told that a string needs to be less than 10 characters, we would test the string at 9 characters, 10 characters and 11 characters to hit the maximum boundary and the closest number above/below it. Acceptance testing is testing the code to ensure it meets customer requirements, which was the whole basis of our project. This was performed subsequently with unit testing, in which we tested each feature’s individual components against the requirements given.

Other types of testing that could have been useful that I did not implement are things like system testing and integration testing. If we pushed this project further, these tests would be crucial to the development of it. System testing tests the overall system working as one program, to ensure everything works properly. In our current case, it is not really doable because we don’t have a functioning system. Currently, the separate features do not function together which means that we wouldn’t be able to test the system as a whole. Integration testing is essentially the same thing, but a step below. Integration testing doesn’t have to involve the entire system, it can test two separate classes working together to ensure they function properly, but it can test the integration of all the classes in the system as well. Again, if we had elected to move this project further, both of these kinds of testing would be crucial.

* 1. **Mindset**

The mindset I had while working on this project was honestly all over the place. I had to put myself in the shoes of the customer, the cyber team, the consumer, as well as the future developers. I had to see the product from the customer and consumer eyes to ensure I developed classes that would function in a satisfactory manner, beginning the process to a smooth and overall well performing application. I had to put myself in the cyber team’s shoes for sake of security. Security is crucial in any development with the amount of vulnerabilities and exploits that have been discovered. When crafting each class, I had to choose whether something should be publicly accessible or private which took some research to be accurate with. Seeing this project from a future developers eyes, I made sure my comments were clear and my code was clean. I would not want to be a future developer for this project and have to begin work on code that is unorganized and all over the place.

As a software developer, I think there is clearly huge bias when testing your own code. The main point of bias being simple functions that you are confident you did right. When testing your own code, you may opt to skip testing these simple things for efficiency, but it is still necessary. Another point of bias in testing your own code is not testing certain cases you know you implemented well, whether that is test cases or even positive test cases. There are lots of bias that stems from confidence in your developing ability that can weaken the testing of your own code. For example, when we had our first milestone it never occurred to me to test the positive cases as I just assumed that they worked, and I tested the negative cases anyways. This could lead to a huge problem further down the line, not ensuring that such a basic component functions properly.

**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.**