**Summary**

My unit testing approach aligns with the software requirements in that I used a variety of test cases to ensure that the program functions as it should. Ensuring the software requirements are met by using the unit tests makes it so the customer’s requests are met and the product that ships is what they ultimately wanted. For example, I used a test to make sure that a contact could be added, and then another separate test to make sure a contact could be deleted. Also had tests to check the length of each criteria, contact id, first name, last name, address, and phone number, and made sure they didn’t go past the limits set by requirements.

I believe my unit tests represent high quality. They are descriptive and easy to read. I made sure to use most of the tests on a per function basis. Doing this allows for each test to be given a specific task and to make sure it completes said test. Having individual tests ensures less complexity and easily identify which test does which, as they are named accordingly. For example for the tests where I verify if a task ID is either too long or null or exactly 10 digits, I would create tests for each scenario separately to ensure each test completed on its own without interference or influence from other tests. I also used a coverage tool, and my tests had around 88% coverage rate of the code, indicating the tests were able to effectively cover the majority of the code.

I was able to make sure that my code was technically sound and efficient using a variety of different techniques. I would use the JUnit framework to ensure consistent tests would be performed. I also made sure to use a consistent coding style and naming scheme when creating the tests. Since they all followed the same structure, it made it easier to read and understand what each test was for and made creating the tests efficient since they were all similar in style.

**Reflection**

As with many aspects of life, software testing also has techniques that are better to use in certain situations than others. This is why during my software testing, I didn’t use every available technique to complete each test. I decided which ones were best for my scenario and used those.

One of the techniques I used during my testing is test planning. I used this technique as taking a step back and planning what aspect of the code needs testing can make creating the tests themselves easier and more efficient. For example, with the Contact program, I know that one of the functions was to add and delete a contact from the database, along with updating various records after they have already been entered. Having a test plan allows me to break down each function individually and more specifically, creating tests for each specific function. Another technique I used was unit testing. This builds off of test planning, in which the actual testing takes place. With unit testing, you test individual components of code. For example, with the contact program, you need to make sure that the contact ID can not be more than 10 characters long and cannot be null. So I created tests for each case, one where the character count was over 10, equals 10, under 10, and when it was null. Having tests for each individual case ensure every possible scenario is tested for so everything scenario is accounted for.

One of the techniques that was not used during the testing process was acceptance testing. This form of testing is when the testing is done by the customer or end user of the program. Since this project didn’t necessarily have a customer, this technique was not utilized. Another technique not used is performance testing. As this testing was not for how the program performed but about how it functioned, performance testing was not used either.

Test planning and unit testing will both be practical in many different development projects. Having a plan on what to test and how to test it will make testing more efficient, as you won’t have to figure out what to test, it is already outlined for you. And unit testing helps make sure each piece of the code is tested properly, making sure to find errors as soon as possible to iron them out and correct them. The earlier the errors are caught, the earlier they can be fixed, while also saving costs.

While acting as a software tester, my mindset I adopted was one of caution, as any little mistake could make the entire project unusable. It was important to appreciate the complexity of the code as so many of the different parts worked together, if one failed then it had the chance of effecting everything else. For example, if the section of code that adds a new appointment doesn’t work correctly, then the core function of the program doesn’t work as you can add new appointments, so all the testing of making sure the appointment ID’s and names and so on aren’t past a certain length is for nothing if you can’t add a new appointment to the program.

I tried to limit my biases while developing this program by using plenty of unit tests to make sure the program is behaving as it should. I can definitely see where there could be concern regarding the developers testing their own code, as they can get defensive is something doesn’t work and try to justify their reasoning of keeping it in, or they could possibly not see a particular issue at all, but an outside party might have. Being aware of such biases as a developer doesn’t necessarily erase the biases altogether, but it does help in lessening the impacts of said biases.

The importance of being disciplined in software development is crucial in making good code. Being disciplined allows you to follow correct protocols and not cut corners. Cutting corners with software could mean the difference between someone paying an extra $10 for a muffin, or thousands of computers being infected from malware. You want to always make the best code you can within your time and resource restrictions, as that is a reflection of you and your company.

References:

Hambling, B., Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing an ISTQB-BCS certified tester foundation guide - 4th edition*. BCS, The Chartered Institute for IT.

Garcia, B. (2017). *Mastering software testing with junit 5*. Packt Publishing.