The main goal was to have the Junit tests be effective and for the code to meet all the requirements. In all the projects including Contacts, Tasks, and Appointments, you had to make sure every variable was not null and fell within a specific length for a string or int variable and also use a THIS statement to make the values changeable. You also need to first call the variable by a string or int variable then use a getter statement to return the variable. The way I made sure it was aligned with the software requirements was making a quick checklist with all the necessary directions to ensure I have met all the necessary goals. In all the projects including ContactService, TaskService, and AppointmentService, we need to make sure there is a connection between the ContactService and the Contact class via a package to allow the variables to successfully transfer over to the ContactService class. For all the classes, we need to have a counter to show how many contacts, tasks, or appointments there are to ensure when any of them need deleted, there is at least one to delete to avoid an error. When we add a contact, we also need to ensure all data added for a new contact is up to code and does not have any errors. You will also need to import Java.util.Random to allow for a unique id to be presented for each new contact. For the ContactServiceTest, TaskServiceTest, and AppointmentServiceTest, you will do a test to ensure that working data will properly go through and any error catchers will properly get anything that is off.

I know the Junit tests were accurate because they met the 80% coverage. Throughout the three assignments, most of the code stayed the same with some changes so with the first Junit test going well helped with the other tests going well. I knew they were effective because all of them were above the 80% coverage range.

One of the ways I determined the code was sound by using test cases to determine the code did what is needed and make sure all the error catchers did their job when an error was entered.

One of the ways I ensured the code was efficient was looking for code that did not need to be added and avoiding redundant code helped keep the code simple and more organized.

I employed caution by making sure all of the code was good step by step before I went on to the next direction to ensure any mistakes would be seen first by me and not when there is errors in the code later on. One of the complexities was trying to get the variables at first from the contact to the contactservice class but was fixed by putting them in the same package.

One of the ways I reduced bias was by testing from every angle. Doing so was crucial in reducing the amount of error that happened. Bias can be a concern since there would need to always be someone else to look at your code if you cannot properly test it from a fair standpoint.

It is important to have discipline when coding because there are times when a problem is difficult to figure out and if you give up and lose confidence then it will stick with you. Cutting corners is not great because it puts you in a mindset where it is okay and I learned that the hard way when I first started college because I would just look up the answer whenever I could not figure it out at first and instead of going back to the book or trying different avenues, I took the shortcut and I stopped doing it after I realized that I was struggling in the next class up for java and helped me with trying to figure out the solution on my own instead of looking it up. A way I will avoid technical debt is to try simple yet helpful problems from different languages to keep a sharp mind.