My approach for designing the unit test was largely following the Arrange, Act, and Assert methodology(AAA). By following this paradigm, I was able to properly verify the requirements for each aspect of the application code. For example, for each function of the program I would set up the resources in memory (Arrange). Next, I would perform some operation that involved the aspect of the software that was being tested(Assert). Finally, I asserted that the operation worked as intended for the test case(Assert).

My unit test properly ensures the functional and nonfunctional requirements for the app by organizing each test to correspond with a desired aspect of the app. For Example, if arguments for a function required input validation that needed to fit a certain criterion, I implemented a unit test that verifies that it works properly when given valid arguments. Additionally, there would be another unit test that verified that the proper exceptions were thrown when invalid arguments were provided. My approach to verifying that the solution worked for the application requirements was pragmatic at its core. By using the AAA methodology described above I was able to design and implement a test for each requirement as needed.

Writing unit test for project one showed me the power of testing software during and alongside development. The ability to write good unit test for the application allowed me to use more advanced programming techniques such as incremental and test-driven development. Test-Driven development played a big role in the quality I was able to produce for the code. For example, designing the unit test for contact service helped me identify any additional methods we would need for testing, methods that were not in the app requirements but were needed to test the code.

The only software testing technique that I used for this project was the AAA (Arrange, Act, and Assert) approach to developing unit test. At first the AAA was only a natural approach to verifying the functionality of the application and I had done little research or even knew that it was the testing approach I was using. It was only after reviewing feedback on my task service application that I understood that it was a well-documented testing approach that seemed to work well for this project. The first part of each unit test set up the resources for testing such as initializing objects declaring variables as needed. Next, I performed some operations on the resources that were specific to the aspect of the application that was being tested. Finally using the JUNIT framework, we make assertions about the qualities that were edited in the earlier step. Declaring variables, performing operations, and asserting that the desired qualities were found in the resources after was the foundation of most of my JUnit testing for this project and I will provide an example below.

A screenshot of a computer

Description automatically generated

My original approach to this project was admittedly slopy. I started by trying to create a side executable file with its own main method that would just test all characteristics of the application at once but after reading the required resources for the modules I learned the power of Unit testing. By writing a small amount of code, designing a corresponding JUnit test, and ensuring that the code passed the test I was able to write some good quality solutions. In particular I enjoyed knowing what part of the JUnit test failed due to the easy-to-understand stack trace that allowed me to trouble shoot bugs faster than ever. Over time I came to respect the quality that can be provided by taking a little time to write proper tests that can build upon each other.

Bias in software development can present a problem when testing code especially in a professional environment, nobody wants to be the guy or gal that sends in bad software and sure we would all like to see the green bubble that says everything is fine, but it is important we keep quality in mind. I can’t speak for all my classmates but managing bias in this project for me came easy because it’s a class project in a learning environment. In a production environment I do think it is necessary to separate the coder from the tester to some extent but not all the time, more of something to consider.

It is important for software developers to commit to providing quality code because of the implications of some of the software we right. Medical, Industrial, and Recreational industries all use software and faulty software development, deployment, and testing can lead to real life fatalities, businesses failing, and privacy violations so it’s important that we take our job seriously. As a former rigger, in charge of making parachutes for soldiers deploying from planes, I understand this importance all too well. This is a serious business and we riggers have a saying we used to keep close to heart "I will always be sure".