Before starting last month I had never heard about most of the different testing that was out there for software development. Going through these last two months it has opened my eyes to something that is so important to every program and software that I will create. I started out thinking that the only testing that I had to do was look over my code and if something was wrong putting print statements would be the best way to find out. Now that I have some experience with JUnit testing I see how essential a tool like that is to make sure that my programs output what their supposed to, or throw when they get an invalid input. The whole project was very academically enriching because it led me down a path of studying more and more and helped me think of all the possible ways to test something.

My unit testing approach I feel like improved on each one of the assignments that I did. For the first one it was very new to me and I didn’t know anything about coverage. During that time I thought I had the whole program working through the JUnit tests, as all the tests came back as running. However it wasn’t until the next week and the second assignment that I learned about the JUnit coverage tests and how important they are that I took a look at my first program to see what it was and it was somewhere around sixty percent. I added some more tests to get it above eighty and proceeded on with the second assignment where that to was sitting around eighty. During the next week I did some more digging on the JUnit tests and found out on the side there is a yellow or green mark saying why its not covered and a couple of my tests had one out of four branches covered. This is when I had to think why this was and what was happening. Turns out that I was get the throw at the top when creating the class but even though I tested setting the attribute I never tested what would happen if I set the attribute to something not allowed. That’s it when I tested to make sure all of them would throw which lead to my one hundred percent coverage on all three assignments. As for how it all aligned to the software requirement is that I tested each one of the assignments(tasks, contacts, appointments) to make sure they could add their specific designed feature like the task class adding a new task, contact class adding a new contact, and the appointment class adding a new appointment. Also I had to make sure that we could update them, set them, and provide input validation throughout the whole process which my tests did for all of these situations. I know my tests were effective because of the one hundred percent coverage that I finally achieved after getting all branches covered on every line of code.

To make sure that my code was technically sound I had to think about what the most outlier thing that could happen was and then test for it to happen. I think the most outlier thing that I tested for was making sure that if someone tried to delete something out of the array but the ID didn’t match anything inside the array, then nothing would happen. I did this by adding to the array and seeing the length of the array to make sure it was equal to one. Then, after that I would try to delete the item out of the array with an ID that was not existent, and again make sure the length was equal to one and that the first item in the array had an ID of one still. Also there was updating something that was not there, I did this by adding a new item to the array and trying to update let’s say the date of a ID that is not in the array. For this one I used a for loop to loop through the array to make sure that the date that I tried to update never got inserted into the array. At this time I do believe that these were the most efficient ways that I could do these things. I am looking forward to continue learning JUnit testing and its syntax to see if I can become even more efficient for these tests. At the moment though making it expect a throw when doing a certain action that should throw, or making sure everything got added that I wanted to get added by using equals or true is the best way that I have found. Even the for loop, to me makes sure that I go through every single item in the array to read it and ensures that what I added or didn’t want to add actually happened.  
 The software testing techniques that I used in this project were static testing, white-box testing, and unit-testing. For static testing it was when I was looking over my code to review it making sure that there were no mistakes in let’s say spelling or any logic errors I may be able to spot before moving onto unit testing. I use this for every project or code that I make to see if I can find bugs or misspellings before I even run the code. As for the whole project I applied this by reviewing my code and also the tests to make sure that everything was logically correct and to see if I could spot any errors that I may have made. I used white box testing by examining the internal logic of the code that I had written. Lastly, for each assignment throughout the whole project I used unit testing to test cases such as making a new array entry or making sure I can update/delete something. These were made and tested with the coverage being one hundred percent to make sure that the program that was written actually did what I planned it to do.

Some other forms of testing that I did not use were performance testing, system testing, and acceptance testing. Performance testing is a test that tests for speed and stability of a system. During this type of testing, it helps ensure that the software can perform the way its supposed to even when under less than ideal situations. A couple examples of performance testing is load testing and stress testing. System testing is when you have the whole system put together and you test it as a whole. This makes sure that all the different parts work together and it functions the way that the stakeholders want it to. Lastly, acceptance testing is where you make sure that everything does what its supposed to do and what the customer wanted the system to do in the first place.

All of these testing techniques are very important when it comes to creating a successful SDLC. They all are set up so that the end product comes out working and functional. If you skip one of them there is a chance that you will get unwanted results from the system, if it even works at all. Take for example static testing, if you don’t perform the static testing to your code then it could have typos in places that would make it hard to find and cause errors in testing later. Another example would be if you were to skip unit testing and go straight into integration testing, this may lead to many errors that would be difficult to point to and fix. On top of that skipping steps would cause the software to not only perform in adverse ways but lots of money would need to be put towards fixing these problems since the errors and bugs would be moved further down the line in the SDLC causing the money to go up as well. Each one of these tests have their own time and place to be useful to make sure that the system completes and does all that it is required to do.

My mindset changed when I was coding to when I was trying to think of the tests that would cover the whole code. I feel like I went from thinking “ok how can I make this work” to “ok how can this all go wrong”. Thinking about everything that could cause the system to fail or have weird outcomes was another fun part of the problem solving journey you go on as a programmer and I am excited to continue to learn all about it. I think you always have to introduce caution when it comes to creating tests because just like your code, your tests could also not be logically set up correctly and not be able to do what you intended them to do which means its not actually testing what you think it is. For me I had to change a couple of things in the tests when they were not set up and I spent some time looking through my code but it ended up being a typo in the date I put in the test itself that was the issue. I try to limit bias by thinking that I know my code may not be the most efficient way to get everything done but I want it to be, so learning new ways is always important in a career where you have to learn new things every day. I can imagine bias can be a concern because no one wants their work to not do what they intended, but I feel like those moments are the best learning opportunities for a developer to have and grow from. Discipline is a big part of making quality software because staying course and finding the best way to make a program is always better than cutting corners. If you are cutting corners you are not only hurting the software, the stakeholders, but also you are hurting yourself in a big way. As a developer you should always put your best into every software and system you build by following good coding practices and never cut corners on the tests that need to be performed. If not you code not only could become buggy but also can become a security risk as well.

As a developer I want to put my best into every system I ever build and continue to learn the best, most secure, and efficient ways to create systems for stakeholders. I want to be on the never ending journey of learning and becoming the best I can be. I am excited for what the future holds especially after I find new ways that make my code work better and have a better chance of becoming less buggy such as this semester learning about Junit testing.