Steven Colley

Project 2

During the past three weeks I was asked to develop three classes that would help improve my ability to develop error free code along with engaging in the correct mind set for using the unit method approach. The three classes were Contact, Task and Appointment Services; Along with three Junit classes that would demonstrate the effectiveness of all three classes. This report will be a description as well as a reflection of my time developing all three classes and the ways in which I exercised best practices.

First for the description of the classes. My approach was to first read in full what was being asked for the classes. Each one of them requested the development of setters and getters that would validate information as well as return what that information was. I then looked at each section of the asks. For instance, the Contact class required a contact ID that consisted of 10 characters in length and must not be null. I fully developed the functions for the requirements then tested within my Junit test class if all requirements were satisfied. Once I was assured that the function was correct, I then started developing the rest of the functions. I repeated this until all “asks” were delivered. Finally, I made my constructor to bring everything together. Up until that point I modified the constructor to only take in one setter at a time to make sure each setter was correct. In the Junit class I made three different instances of the Contact classes with two expected outcomes. The first would be the data that would be passed and should be returned as what was entered. To do this I made 5 functions to test what was being returned and added an assert if true function for what was entered. If all tests came back as true, then I know the setters were properly setting the correct data. The second and third instance of the classes would only be passes data that should not be accepted. The second instance was passed data that was deemed too long for the setter to take in. The third instance of the class would only be testing if null was stored. For the second instance I tested it like the first instance but adding an assert if false for the data that was being entered. This is because if a firstName variable was passed that was too long and the getter didn’t return that variable then I knew the setter wasn’t accepting the variable being passed. Finally for the third-class instance I created tests that would make sure the data wasn’t null. Like the other tested I made an assert if not null. It the tests passed then I know the variables weren’t null. To test the corresponding contact service class, I only had to make two different tests that would check if my vector of contact class stored data and deleted data. To do this I added an instance of the contact class to a vector and added an assert true if vector size equaled one. I tested if the classes were deleted if once delete class method was called the vector size was zero. I did this for all the classes that were asked for and after all tests came back true, I knew all classes were developed properly and ready for submission.

In this section I will reflect on my experience on using different methods of testing and describe my experience. Thankfully the assignment required us to use both static and dynamic testing for both classes. Static testing happened during the building of the contact class and dynamic testing happened during the development of the contact services class. During the contact class I tested each setter function before moving onto the next setter and before completing the constructor. This allowed me to be ensured each setter was complete and when it came time to create the constructor, I could focus on ensuring that if a problem arose then it was for my logic of the constructor and not the previously developed setters. The dynamic testing was used for the contact services class because everything had to be developed first so I could store that information into a vector of contact class. The benefit of completing the contact class first was that I knew the class wouldn’t terminate because information wasn’t passing through the setter functions properly and it was indeed the way I was adding a class to my vector. Other methods that were used while creating the classes was the white box testing because I knew what data was being entered and how that data was being stored within the classes. Unit testing was also used by creating each setter first and calling a getter and confirming the data was correct. Two of the testing methods I didn’t use was the regression and black box testing. This is because I wasn’t dealing with old code or old data so I had no ability to regression test, and I also couldn’t black box testing because I knew the data along with the workings of the functions. Although now that I think about it, I could try a form of black box testing by adding a random number generator that also randomly adds character to a string and passes that information to the constructor/setters.

In this section I will be reflecting on the mindset of testing. I think the most important thing to think of when tackling any problem or test is to just start and then proceed with one step at a time. Unit testing allows for the developer to not be fully aware of how a final product may turn out in terms of errors and still start creating correct code. For instance, when making the services class for all three of the assignments I knew what the services would need but not what each of the parent class of those services classes. The unit testing allowed me to feel confident that I was one step closer to be complete without being concerned that I would have a problem with my code down the development road. Developing this way also allowed me to focus on what was wrong with my code when an error did come up. This is because I had already confirmed that other parts of my code were correct and didn’t need to waste time on what I knew wasn’t the problem.

I didn’t really think about biases within my code before probably because I’m not that confident in my coding abilities to begin with. I feel like I need to always test and retest everything to minimize mistakes. This is just a lack of experience on my part. However, I can see how more experienced programmers can become biased towards there own code and how unit testing can make them see the flaws that have been created because of poor code. More than likely, it would come from an oversight of what is needed within their code or accidentally flipping variable locations that would need to error. Unit testing allows developers to hold themselves accountable to themselves. When developing code, it is important to not cut corners because this leads to oversights that could be hard to debug later, debugging that may not be done by the original developer. If we as programmers start to get lazy then money and time will be wasted down the road. If programmers get lazy while testing and we can not properly validate our code then we cannot with 100% certainty, at least honestly, endorse our work.