Project 2

There were a few things that I did that constituted my approach to ensuring my software aligned with the functional requirements. The first thing I did was to check over the requirements themselves and what was needed to program each class. First, I built the basics of each class, such as the attributes required and all of the basic functions like getters and setters. Next, I went back to the requirements. As an example, the first name requirement to be 10 letters or less. I started by determining what tests needed to be performed to ensure that my code would align with the requirement. I figured out that I needed to do boundary testing, and test below, at, and above the limit. The name also could not be blank or null, so I made sure I had tests for those as well. I now had 5 things to test for and went about creating the tests in the test class. First, I just made a function for each test. No code yet, but just a name that made it obvious what each test was testing. I did this for each attribute and its requirements as defined in the assignment. Then I went back and started writing the code for the class, keeping in mind the requirements. After that, and once the code was complete, I went back and finished writing the tests with the functions that I had created. After running the unit test, I was able to look at the coverage and determine that my tests were successful and that I had covered at least 80% of the class being tested. While ensuring my code was technically sound, I made tests for each requirement. In the ContactTest class, at lines 8, 15, 22, 30, and 38, I used 5 different tests. Three were to test each part of the boundary, and the others for null and blank. To ensure code efficiency, I made sure to make my setters handle any errors and to use them in the constructors. This way, I didn’t have to reuse the same exception handling code multiple times. I used the same techniques for the appointment and contact classes. Other requirements, such as the unique ID that couldn’t be updated or duplicated, I decided to set that on my own with the code. This way, the program would create unique IDs for each object up to 9,999,999,999 different objects. While this still has a limit, the likelihood that someone knows or has more contacts than there are people in the world is small. If the customer needed to change how the unique IDs were used for some reason, this could always be changed later on.

Some of the specific software testing techniques that I used were unit testing, functional testing, manual code review, and integration testing at a small scale. Unit testing involved writing the different unit tests for each class to ensure that the functions worked as intended, such as not allowing a first name for a contact to be longer than 10 characters. All of the testing was functional testing since the purpose of the testing was to make sure that the functional requirements of each class had been met. While coding, I made sure to manually look over my code as well. Just because a test passed, doesn’t mean that it didn’t potentially pass by accident. For example, a test of a function to add two numbers might pass if you used 2 and 2 for your numbers. However, if your function is multiplying the two numbers instead of adding, your test would still pass. So, making sure that both your functions and tests look correct and are designed in such a way as to leave no doubt that they are working as intended is important.

I didn’t do any performance testing. Not only did we not have any kind of nonfunctional requirements, such as those related to performance, but our software was not at a level of completion that those tests would have had any real impact or meaning. Also, since we didn’t have an interface of any kind, we couldn’t do any black box or exploratory testing. Both of those tests would have involved someone testing the software without knowing how it was coded. Essentially, it would be like giving a calculator to someone and telling them to make sure that it can give you the correct answer to anything you punch into it. Your calculator might do the multiplication of 8 times 8, by adding 8 to itself 7 times, or whatever, but it doesn’t matter as long as the answer is correct. Practically, the tests we used are good to ensure your software is doing what the customer asked for. If your code can’t pass a functional unit test, you aren’t doing the job you were hired to do.

The mindset for testing was just as important as the knowledge of techniques. Being cautious and meticulous is just as important in determining what and how to test. Just because you write a few tests does not mean you tested everything or that your tests themselves were written correctly. Making sure that you know what it is you are testing, what kind of testing you should use, and what the expected results are, are all important. In order to eliminate bias in my testing. I tried to come up with the appropriate tests before writing the code. This way I can code based on an expected outcome, as opposed to writing a test that I know will pass based on my code. Being disciplined is just as important. We have all heard of ways in which bugs in code have caused massive issues. No matter how insignificant a bug, it can potentially be exploited to cause harm, or unknowingly cause harm. I like to think of it like this… If I am coding the engine control model for a car, I want to test the code as if my wife or kids drove that car. If I don’t properly test it for bugs and something bad were to come of it, my own family could die from my negligence. If that means that my code comes out twice as slow because I create a unit test for a class or a function as soon as I finish creating it, then that is how it works.