Jacob Feldman

2/17/2023

CS 320 Project 2

**Summary:**

The way in which I approached designing the mobile application was to test it strenuously and often. By incorporating software security requirements, I was able to ensure that the application was as safe and secure as I possibly could. For example, the contact class has a parameter where the characters making up the user’s first and last name cannot be empty nor greater in size than 10. This is in direct guidance of standard practices. The ContactTest class was created to ensure that the character requirements were met. The JUnit test will verify that the input is between 1-10 characters in length. If an input is given that doesn’t meet the parameters, a JUnit test incorporated into the TaskTest class would result in an argument. I found that the JUnit tests were successful based on the coverage percentages being quite low overall. Just receiving a coverage percentage is a sign that the tests were running correctly. The resulting low percentages indicate a strong coverage and low risk.

To ensure that my code was both technically sound and efficient, I made sure to review the supporting resources very carefully. The textbook provided this term has been tremendously helpful in learning about software security and improving my coding practice. I learned that incorporating strings and arrays into my code was the best way to make it more sound. For example, in the contact class again, I created a list for all of the strings. This line of code makes JUnit testing easier to execute because adding/removing items from the list is a simple task.

“public static List CONTACT = new ArrayList();”

Coding efficiency is learned through practice and studying. I am still a newer programmer so I am not yet gleaming with confidence, but I feel that I am learning quickly. I am lucky to have a few programmers in my extended family that can try to help me when I’m struggling. YouTube has also been helpful, but not as much as I’d hoped it would be. When I think of efficiency, I not only think of how quickly the program will run, but also how easy the code would be to read from another perspective. This is why I tried to define everything clearly in the contact class. Each variable is defined separately so that it is obvious as to what the code is saying.

“private String contactID;

private String firstName;

private String lastName;” etc.

**Reflection:**

There were a few different software testing techniques that I employed during Project 1. The first and most abundant was JUnit testing. This was used to ensure that inputs were acceptable and that there were little to no vulnerabilities. I used a JUnit test to ensure that inputs for first and last names were between 1-10 characters in length. This is according to the requirements and helps limit potential login vulnerabilities. I also used white box testing for this application. White box testing checks logical and structural aspect of the code and is often used by programmers and testers. It focuses on the inner workings of a software application and looks for foundational vulnerabilities. I employed this testing method so that it would test each line of code individually before moving on the ensure everything worked properly, and to prevent moving forward without resolving any errors.

A few software testing techniques that I did not use were black box testing and static testing. Black box testing is examining an application from a user interface standpoint. It doesn’t focus on the code at all, it just makes sure that the end result is what it should be. This was not quite useful in Project 1 because I developed the application without the intentions of using the mobile application. A good example of black box testing would be if you were a tester for a company who made a new website. Your job is to view the website and make sure it functions as intended. When you click the home button, the website will take you to the home page etc. I did not use static testing either because it requires that you do not execute any code when looking for defects. Since I wrote the application, dynamic testing techniques were appropriate, such as JUnit testing and white box testing. An example of static testing would be a code review. When developing pseudocode, you have an excellent opportunity to edit and revise the code without executing it. This is very useful during early stages of the software development life cycle.

**Mindset:**

Software security measure were quite foreign to me before I began this class. Now that I have some experience with it, I have started to look at the way I approach projects differently. I now have security and efficiency at the forefront of my priorities. Caution was certainly in my approach for Project 1, but I was more focused on learning the specifics of the testing techniques I was using. I found that being patient is the most important part of this experience. It can be frustrating when things aren’t working and you can’t figure out why. This gave me an appreciation of how difficult and complex this kind of work can be, and ultimately how important it is. Nearly every day in the news we hear about companies being hacked and millions of people losing their private information. I never really thought about it much until I had my identity stolen a few years ago. It is one of the reasons I am here at SNHU and taking this course right now.

I haven’t experienced much bias personally when writing code as I feel that I am still trying to figure everything out. However, I can see how bias can be an important issue for developers. Coders might feel that the way they do things are the best even when presented with a better alternative. It can be an ego problem just like in sports or politics. Even their personal biases can negatively impact their work. For example, a programmer might be making a website where people are supposed to view political news in a non-biased manner. If the developers have political biases, they can promote the news that they like and hide the news they don’t. To be fair, this example is only really an issue for news organizations that promote themselves as unbiased.

Discipline and patience are essential to be a successful programmer. This means no cutting corners and no lazy alternatives. Being meticulous is vital to ensuring that your code is sound and properly secure. Just like any major problem, they stem from something smaller and more preventable in the past. Lazily or not implementing security measures at all will result in a poor final product and an unsatisfied user base, as well as putting their information at greater risk. It is crucial to test and retest your code before implementation no matter how tedious it may be. I plan to avoid this type of issue in my future endeavors by mastering software security measures. Once I am very confident in my abilities, I will have no problem sitting down and spending the time necessary to do it right. Whatever application I may work on or project I may assist on, I am going to make sure that whatever product I put forward is the best that I can possibly make it. Most software security issues seem to stem from cutting corners or rushed deadlines. If it were up to me, I would never release a product that I wasn’t 100% confident in, regardless of the time or budgetary constraints.