Dylan Ackron

CS 320

SNHU

June 17th, 2024

When developing my mobile app milestones, my testing approach was to make sure that every line of code that has some sort of action or input was thoroughly tested and not missed during the coding of my test cases, because one small miss could be impactful. I ensured that I followed the requirements closely. Ways I did this for example was that each string was required to be within a certain character limit or even an exact character limit, and while coding the application those requirements were incorporated exactly as the requirements specified. In my JUnit testing, each of those requirements were also tested to ensure they were functioning properly to make sure the requirements were going to be met. I believe the quality of my JUnit testing was good, especially considering this was my first time doing it. During the creation of test cases I tried to test every line of code that had an action, such as the creation of strings and other variables. My coverage percentage on my JUnit tests were almost all 100%, making sure that instructions weren't missed during the testing.

This class was my first interaction with writing JUnit test cases, or any sort of test cases for that matter. One way I made sure my code was technically sound was when a user would want to add an appointment, they would have to create a unique ID, and if it wasnt unique the program would throw an exception. “public void addAppointment(Appointment appointment){ if (appointments.containsKey(appointment.getId())) {

throw new IllegalArgumentException("Appointment ID must be unique.")

}”

This is just one thing of many that could be checked in the programs to ensure that it was technically sound. While coding I tried to be efficient with my “IF” statements. I knew that if there was room to just fit multiple into one rather than needing to create multiple, it would allow my code to look much cleaner and be more readable. One example is “if (description == null || description.length()>50) {

throw new IllegalArgumentException("invalid description");”

The only other type of testing I did within this project was integration testing. Instead of coding everything and then just testing everything at the end, I would write a block of code and then test it then and there rather than getting down the line and needing to change multiple things at once to get everything working properly. Types of software testing techniques that I did not use during this project were things like load testing, this wasn't possible considering I am the only person using the application, so testing it with multiple people wasn't possible. Another method not used was stress testing, for similar reasons. Load and stress testing are two important methods for applications that will have a larger use. These two are important to consider during the development process, because a well working application is useless if no one can use it because it fails under load.

When testing, taking caution wasn’t really something that was running through my mind, I wanted to ensure that all requirements were met so I wanted to test everything I could think of. It's important to appreciate the complexity and interrelationships of the code because I wanted everything to flow correctly and work cohesively. One example of that was figuring out how to properly implement the date function in the appointment milestone. During testing, ways I tried to eliminate bias was kind of straightforward for me, my goal was to test all the code I can think of, and if it simply doesn't work as intended, fix it until it does. However, I can see that bias may come into play, because maybe the person who coded the program isn't trying hard enough to “break” the program or get it to show faults. Whereas an actual tester may push it to its limits to ensure everything meets requirements and then some. A developer may create test codes to closely related to what they’re testing, leaving no room for other possibilities, creating a possible bias in testing their own code. Being disciplined while writing code can save a lot of time in the long run, cutting corners during the development and testing can cause faults to be missed. So producing quality work the first time around may save time and money in the future. To avoid technical debt I simply plan to take my time to ensure quality work is produced the first time around, rather than speed running a development and test process.