Brent Colyer

October 13, 2022

Instructor Jacks

CS-320 Software Testing and QA

While developing the contact, task, and appointment services for the mobile application, testing was at the forefront of my mind. Every piece of code that was written needed to be tested to ensure that it was aligned with the software requirements. For each software requirement, a method or line of code was written to ensure that the requirement was included in the application. Then to ensure that the code written for the requirement was correct, a JUnit test was written. This strategy made sure that every software requirement was addressed and then tested to ensure that it was correct. For example, in the Contact Service java file, there are methods to add contacts, delete contacts, and update the first name, last name, phone, and address of a contact. Then to ensure that each one of these methods was correct a JUnit test was written for the add contact method, deleted method, and update first name, last name, phone, and address. In addition, the JUnit tests were effective because each one of the files achieved 98-100% coverage. This means that every line of code was tested and passed the test therefore each line of code was technically sound. Within each JUnit test, certain assertions were made to make sure the code performed correctly. If one method depended on the output of another method, both methods were checked even if the first method has already been tested. This ensured that using methods inside of other methods still produced the correct output. For example, in the Task Service java file to use the delete task method a task must be added so before the delete method was tested in the delete task JUnit test a task was added and the add task method was tested to ensure that it worked even though the add task method had previously been tested. This way of testing however did not cause the code to inefficient. The code for each service was efficient and the JUnit tests were efficient as well. No line was code was written without the software requirements in mind and each line has a purpose. This efficiency is clear to see in the Task Service java file as the code is very compact and neat, yet the code still meets all the requirements.

There were various software techniques employed throughout the entire project. All the types of techniques were black-box or specification-based techniques. This means that the JUnit tests were written knowing exactly how the system works internally. The testing techniques did not define how code was written like a model would, but rather tested the code to verify if performs correctly. A type of this used in this application is called boundary analysis which is the testing around the boundary of values that the system handles. For example, in the contact, task, and appointment objects the Id value for each of them was to be less than 10 so using the boundary analysis technique object ID values that were 9, 10, and 11 characters long were tested. This ensured that the object Ids were correctly meeting software requirements and not passing tests by chance. Some of the software testing techniques that were not used in this project were white-box or structured-based techniques. These techniques include using flowcharts and decision tables to determine what the output of a system is without knowing the interworking of the code. Flowcharts and decision tables are usually created before any coding takes place and my first job was to start coding so, therefore, no flow charts were created. Even though structured-based techniques were not used during this project they are important to the successful planning of any software project. Being able to show how a system works before any code is written will help the developers know how they need to write their code to meet certain software requirements. It will also allow stakeholders and managers to visually see how the system is working even if they do not understand how to read code. Then by using specification-base techniques the actual code that was written based on the diagrams can be tested to ensure that it works how it is intended. The two types of testing techniques can work together to make sure that the software requirements are met in any type of software project.

While working on this project, I had to have a different mindset depending on the task that I was trying to complete. When writing code, I was focused on being able to take the code work based on the requirements that I was given. Then while I was testing the code I just wrote I had to think about how to “break” my code. I had to think about where the weaknesses in my code were and how could I test those weaknesses. Knowing that for each milestone two files relied on each other made it even more important to test each line of code. Appreciating the complexity and interrelationships of the code was important because I knew I had to write correct tests week. I knew that if one method in one week was incorrect then other methods that depended on it later would also be incorrect. This made me focus on writing correct tests from the beginning. Limiting bias is also important to remember when writing code because there is more than one way to do things. Just because I did it one way and someone else did it another does not mean one of us is wrong. I think this is where bias would come into play from a software developer’s side. If I am testing my code I might be more inclined to not test as deeply to make sure it looks like my way of coding was correct, but this is not good because it would result in weak code being moved on to new stages. It is also important to test code with the mindset of how I can break this now, so it does not get broken later no matter who the developer was. Being disciplined as a software developer is another thing that might not get talked about enough. Making sure that every piece of code is not only written correctly but testing throughout is time-consuming. It may seem like it would be good to cut corners and save time in the development process, but this could be costly later. The longer it takes to find an error in code the more expensive the error becomes. It is also important to do things correctly the first time and not cut corners because quality should always be the most important thing in any software project. It cost a lot of money to have software developed and the highest quality product is expected by the customer. If cutting corners allows for faster release of a product and the product fails only a short time after being released because of something not being tested the corners that were cut will surely be found. This look reflects negatively on you as a software developer and your company for allowing an untested piece of software to be released. Software should always be tested fully and correctly to ensure that the highest quality product is being released.