Project Two: Summary and Reflection

Rushil Patel

This was my first experience with testing other than a manual check to search for trivial faults. Learning about Junit tests and how to deploy them has taken my understanding of testing and how code works to a whole new level. Being able to write tests for my own code helps me to know exactly what to test for and verify that the code is fluent. When the criteria for the first assignment, Contact Service, were delivered, the code was quite easy. To accomplish that, I was able to construct protected variables using getter and setter methods. We were able to confirm that the variables satisfied the length and non-null criteria using the setter methods. This is also true for ContactService.java, where we wrote code to add, remove, and update each component of the contact. This assignment had the most requirements of the three project one tasks, however it was not difficult to write the code. Every criteria for the Contact Service was satisfied, making it simple to alter the code to meet Task and Appointment needs. For me, the most challenging element was creating the unit tests for the first time because I had never done it before. Knowing what to test for was really helpful, however the actual syntax of the tests created some confusion at first. When I ran the Coverage Junit test, I was able to get the test to execute without issue and get 100% coverage. Although this does not imply that the project is flawless, the unit tests were ideal for the assignment's objectives.

After briefly mentioning the difficulty of developing Junit tests, I will go into further detail below. Because I had never written exams before, I wanted to make certain that I spent adequate time studying the intricacies of these standards. Earlier in the course, there was an announcement post that essentially suggested to keep things simple and just hit prerequisites, which helped me assimilate the material better to fit what we were learning and not go above and above so that the program was not confused.

To explain good and efficient code, I'll use Task.Java and TaskService.Java. This assignment's code was designed to be basic, easy to alter, and test. In Task.Java, for example, I wrote setter methods to ensure that each module verified input. This made it easy to read while also ensuring that each variable was checked appropriately. Furthermore, because I can invoke each method independently, it simplifies testing. Another example is in TaskService.java, where I made different methods to update the task produced rather than constructing a method with lines of code (Lines 47-75).

It may add a few lines of code, but it assures that each method does precisely what it is intended to do.

The subject of efficient code comes next. Despite the fact that there were more lines owing to the number of methods, they are basic methods that can be moved and updated to new assignments with different variable names, which is useful when working on projects. There is no clutter by making explicit comments on what each person accomplishes. It also saves time by requiring only a single test for the technique.

I only used two sorts of tests for the assignments and project. The first was unit testing with Junit tests, which I personally tested by going through each line. Reading it over guaranteed that the project was clean, and building testing classes allowed me to back up my conclusions. Combining these two is ideal since it creates a solid basis for long-term and clean code. Following unit testing, I utilized the Acceptance and System testing techniques to check that all criteria were satisfied and the code was working. Even though I developed each line of code to meet the specifications, checking that the specifications are satisfied saves time. We don't have to go back and construct or alter something to meet the specifications.

Even if our strategies were ideal for this task, there are others that might and should have been employed for a larger-scale application. Integration testing is one example. Because the programs generated were so simple, there was no need to completely employ this method. It didn't make sense to spend time on this because the project didn't have many moving parts. Performance testing is another example of testing that was not carried out. Another way for a bigger project that does more than just accept an input and set a variable. Because it is basic and does not deal with big volumes of data input or code running at once, our software should not stress a machine or the application. This implies that no stress or spike testing are required. The last approach I'll mention that wasn't implemented is security testing. Again, because the software was easy to develop, it was easy to defend. I took care to private variables that were utilized, which adds encapsulation and protection, but the program itself was not sophisticated enough to need running a security test or developing degrees of security. If I had been obliged to, or if it was to be made public, I would have incorporated extra security measures and tested them.

To be honest, switching from developer to tester was difficult at first. I prefer to work in groups, so transitioning from writing to testing was challenging. I found it easier to build the software, take a break, and then return to the project to write the tests. This helped me to be more cautious while writing tests, ensuring that I was not just testing the code, but also the requirements. It is easy to get lost and develop tests that match the code, but to truly test a software, you must think differently. In one of my test classes, I developed a test to determine whether I could add a contact with the same ID as another contact. If I was only looking for a checkmark, I could have developed a test to add a contact, but instead I sat back and considered the needs.

Another issue I discovered was partiality. Because I needed my code to work and my test to verify it was correct, the initial test I created was insufficient. I was more concerned with passing than with meeting standards. Because I slipped into the similar mistake, I can understand how this may be an issue for folks who are new to testing their code. Writing a test that gives you a confirmation bias indicating your project is correct can lead to a slew of security and program integrity issues.

When looking at the overall picture of a project, testing is without a doubt the most critical component of software development and engineering. This is why being diligent in your testing is so important. While the code is required to operate the application, testing will provide the ultimate approval. Not only is testing performed when the project is first established, but a good software is tested on a regular basis while it is in use. Testing guarantees that the application operates smoothly, safeguards data, and leaves no space for a hacker to enter. The software will be as safe and secure as possible from the start, thanks to manual, functional, and non-functional testing.