Jaime Rowland

Aug 8th 2022

Project 2

**Summary:**

My strategy was completely in accordance with the software requirements since I gave it the highest priority to write a code with dependable tests that satisfied each requirement specified. Let's examine the contact class. The first and last name cannot be null, and therefore can be no more than 10 characters in the character value, according to the standards. In our ContactTest class, the JUnit test had an assertion to see whether the input was too long. According to the requirements of the task class, the task ID/unique ID could not exceed 10 characters. Since the task ID may be more than 10 characters, we used a function in our TaskTest JUnit test that would throw an argument flag.

The resources in each module greatly contributed to raising the coverage percentage, which resulted in an overall improvement in the quality of my JUnit tests week by week. In comparison to the coverage percentage in the contact test, the task test had a relatively low coverage percentage, and the service test had the highest total coverage percentage. Any time a positive coverage % was shown, I knew that the JUnit tests I was running had successfully covered most of the code's functions.

I utilized both strings and arrays to make sure my code was technically sound. As an illustration, consider the contact class. To keep track of the contacts, I created a list for the strings. An example of this is: “public static List CONTACT = new ArrayList();”. In turn, this would make it simpler to execute my JUnit tests and add to or remove from the list.

I took various steps to make sure my code was effective. I have some basic knowledge of coding from earlier classes I took at Southern New Hampshire University, but whenever I feel like I could use a refresher, I turn to YouTube mostly. YouTube has a wide variety of videos would there to help. I also believe in running my code frequently to make sure there aren't too many problems as I go, which will help to ensure that my code is effective. An almost trial-and-error procedure. Additionally, I made care to declare each variable before using it. As an illustration, consider the Contact class. I disclosed all of the variables that made up a single contact before comparing it to the required specifications.

**Reflection:**

Every project must have testing to be successful. I used a highly detailed strategy when using software testing methodologies for this project. I felt like I had a test for everything as we went over the requirements, such as how many characters a first and last name can have and that neither name can be null. I took care to name my tests in the JUnit tests in a way that was quite similar to the names of the core classes. Everything was kept incredibly organized thanks to this. The step-by-step method was another software testing method I employed. Where I made sure I didn't miss anything by going through each criterion one at a time. I then made my test cases after I was certain that each requirement had been recognized (JUnit). To finish any structural and logical testing, I additionally used White Box Testing in addition to JUnit testing.

Static testing is a method of software testing that I did not employ for this project. Static testing involves looking over code without running it to find any issues. I never just let my code be tested visually; I always run it as well. However, I like to click the Build All or Debug buttons for problems to appear. I understand that static testing is focused on discovering issues early as preventive through walkthroughs, code inspections, and peer reviews.

Static testing is a software testing method that I did not use to this project. Static testing is the process of reviewing code without actually running it to find any flaws. I never simply let it be a visual thing; I always run my code to test it. I am aware that the goal of static testing is to identify issues early on and prevent them through peer reviews, code inspections, and walkthroughs, but I like to use Build All or Debug buttons to see if any errors appear.

My perspective has really changed from when I started working on this project. Honestly, I have no real previous knowledge of any kind of software testing prior to starting school. I honestly believed that "testing" referred to just running the program to check for problems at the bottom. Acting as a software tester is something I would define as dedicated rather than cautious. We cannot take any short cuts in order to complete a program more quickly and we naturally want to guarantee that the final output satisfies all client demands we must constantly ensure that the final products quality and functionality are not compromised.

If you wrote and tested your own code, I can understand why someone could suspect there might be a bias. The simplest method to limit bias, in my opinion, would be to always maintain humility and to be open to change and learning. There is no room for improvement or for finding errors if we test our own code, automatically assuming that we are the best programmer in the world. Our final product might suffer as a result of this.

As a professional in software engineering, discipline is crucial to my dedication to excellence. We not only adhere to a strict set of software engineering ethics, but it is also never acceptable to skimp on the creation and testing of code.