CS 320 – Module 7 Project

Matthew Haggard

When developing tests for the final project, I took the experience that I had gained in the first three modules and applied it to the project. As I developed each class function, I would move to the testing class and write a test for the function. This helped me achieve the 80% coverage that was mentioned by the requirements. Also, I made sure to test for the specific asks of the project requirements. For example, in the task class, each of the private variables (taskId, name, and description) had character limits. I ensured the correct number of characters by adding the following code to the constructor:

public Task(String id, String name, String description) {

if (id == null || id.length() > 10) {

throw new IllegalArgumentException("Invalid task ID");

}

if (name == null || name.length()> 20) {

throw new IllegalArgumentException("Invalid name.");

}

if (description == null || description.length() > 50) {

throw new IllegalArgumentException("Invalid description");

}

this.taskId = id;

this.name = name;

this.description = description;

}

Each if-statement in the constructor checks to see if the respective variable is of the appropriate length. Next, I added a test in the TaskTest class to verify that if an invalid number of characters is entered that the IllegalArgumentException is thrown appropriately. I repeated the same style of testing for each variable in that class, along with the other classes of the application. When testing the methods in the respective classes, I used a similar approach to verify their validity. As an example, when testing the “getters” for the Task class, I created a new task object and assigned it valid values. Then, I used the assignTrue() function to test to see if the arguments had been assigned correctly. The following code is an example of that testing method:

void testTaskAndGetters() {

Task Task = new Task("1234567890", "Do laundry", "Do all of the laundry");

assertTrue(Task.getTaskId().equals("1234567890"));

assertTrue(Task.getName().equals("Do laundry"));

assertTrue(Task.getDescription().equals("Do all of the laundry"));

}

The most prominent software testing method I used in this project was white box testing. White box testing is a testing method that is used to test outputs of provided inputs. Some of the benefits of white box testing are that it allows the tester to achieve complete code coverage, it reduces the communication overhead between testers and developers, and that it allows for continuous improvement of code and development practices. (What Is White Box Testing | Types & Techniques for Code Coverage | Imperva, 2020) One of the major flaws of white box testing is the inability to test from the user’s perspective.

One example of a testing method that I didn’t implement in my code was black box testing. Black box testing tests the application as a whole without prior knowledge of its implementation. (McKeever et al., 2020) This style of testing is very beneficial because it allows the tester to test the application the way that a user would test it in a real-world scenario.

When I began developing this project in the module 3, I had maintained my mindset of write code, the write a debug test to ensure that the output is what I expected it to be. That didn’t necessarily mean that the code I was writing was correct, it just did what I thought it should do. Developing applications throughout this course and, more specifically in module 6, allowed me to gain knowledge about how to test functionality of my code with accuracy. It made me more aware of what code I was writing also. In the past, I would write my code much like an amateur cook prepares spaghetti. I would just, metaphorically, throw it against the wall and see if it sticks. If it compiled and ran, I was satisfied. This project has helped me to write code with purpose and more understand how to test the code for errors accurately.

I came into this class with a pretty open mind to learn, so from a bias perspective, I don’t personally feel that I encountered any issues with bias while developing this project as I was legitimately on a mission to learn the material. In projects that I’ve developed in my free time, however, I can attest that there have been occasions that I was too proud to remove something or correct something because of bias. Developing software can be considered an art, much like playing music. When someone is judging their own art, or in this case code, it can lead to overlooking key bugs and deficiencies. That is why it is very important to allow someone with an unbiased perspective to develop and run the tests on the code to ensure those deficiencies are handled properly.

Having a commitment to writing quality software is not only an important piece of someone’s character. Badly developed software can have very real effects on people that use the software. While most impacts of software are positive in nature, a development error that creates a vulnerability can impact people’s financial well-being, employment status, or even cause real physical injury or death. One major example of the impacts of vulnerabilities in software was the PlayStation Network outage of 2011 where around 77 million PlayStation accounts were accessed and were unable to use the service for 23 days. This exploit caused personally identifiable information for those users to be exposed. While there were reports of credit card fraud that came about during the hack, there was no real reported impact to the users’ safety. This event could have been much worse that it was considering the nature of the information released. My plan to avoid events of this magnitude and to maximize the quality of the code I write is to strive to maintain an open mind to the feedback I receive on the code that I write. Also, I plan to always pursue knowledge of new ideas and techniques as the industry evolves.

References

What Is White Box Testing | Types & Techniques for Code Coverage | Imperva. (2020, September 24). Learning Center. Retrieved October 16, 2022, from <https://www.imperva.com/learn/application-security/white-box-testing/>

McKeever, G., McKeever, G., Hewitt, N., Wall, P., Lynch, B. L., Lynch, B., Lynch, B., & Lynch, B. (2020, September 24). *What is Black Box Testing | Techniques & Examples | Imperva*. Learning Center. Retrieved October 16, 2022, from https://www.imperva.com/learn/application-security/black-box-testing/