**7-2 Project Two Submission**

**Jesse Moore**

**CS 320 - Software Test, Automation**

**Professor Tuft**

**Southern New Hampshire University - Computer Science**

**Summary - Unit Testing Approach:**

Starting off, before I began coding the project requested of me I examined the requirements documents. From there I was able to determine not only how the requested software would be built, but also what was required of me in terms of testing. Each function within the software had at least two requirements, both needing to be met in order to fulfill the request.

The software I provided in the end did well to meet the requirements set for me. I ensured that all requested functions were present in the final product and working as intended. The quality of my JUnit tests was decent. My tests were tailored to the requested functions and the bounds set for them by the client. I did my best to ensure that these tests showed the functionality of the software I provided and I did a decent job on that front. One issue with my JUnit tests though was the code coverage. My code coverage score was at 77% which isn’t bad at all, though this does leave some room for improvement to foster more trust in the finished product.

This was my first time writing my own unit tests, and because of that it definitely left room for improvement. While I wrote solid tests, there is still a lot I have to learn in order to better test code I’ve written. The tests I did utilize checked to make sure that values the user may input would remain within predefined bounds, and if they didn’t, other tests would ensure that the proper exceptions would be thrown notifying the user of the issue while preventing a crash. Other tests I utilized ensured that objects created using user input would in fact be created. More features that were tested include the functionality of the update functions and delete functions.

On the front of efficiency I made sure to keep my code short and concise, avoiding both redundancy and unused lines of code. Initially I had some imported test utilities that I did not end up using as I was still feeling my way around unit testing but I made sure to remove them before submitting the final product.

Here are some specific examples:

The testTaskCreation method verifies that the Task object is created correctly and that its fields match the expected values.

@Test

public void testTaskCreation() {

Task tasks = new Task("0001", "Java Assignment", "Create a Task service along with appropriate tests");

assertEquals("0001", tasks.getTaskID());

assertEquals("Java Assignment", tasks.getTaskName());

assertEquals("Create a Task service along with appropriate tests", tasks.getTaskDescription());

}

Another example is here, the Task class validates input parameters immediately in the constructor, preventing the creation of invalid objects and avoiding unnecessary processing.

public Task(String taskID, String taskName, String taskDescription) {

validateTaskID(taskID);

validateTaskName(taskName);

validateTaskDescription(taskDescription);

this.taskID = taskID;

this.taskName = taskName;

this.taskDescription = taskDescription;

}

**Reflection:**

Through the completion of this project I have used quite a few techniques to ensure that the code I provided was up to standard based on the requirements. While implementing unit tests I did my best to make sure requirements were fulfilled by checking that required fields that would be entered by the user could not be updated as requested by the client, fields would not be too long or too short based on specifications, and ensuring the actual creation of an object once called.

Specifically, for the first milestone I utilized assertEquals to check if my proposed input would actually populate as commanded. I also used assertThrows to test the functionality of the throws I put in place to try and catch issues within user input. The next milestone I used similar tactics (assertThrows and assertEquals) but I also used assertNotNull. Something else I utilized here was the @BeforeEach function in order to create a fresh object before each test in order to avoid issues down the line, just to be safe. I used Positive and Negative Testing techniques, State-Based testing, as well as Exception and Boundary testing in order to provide a thoroughly tested product.

For this project I have not used Behavior-Based testing. Behavioral testing would be beneficial for systems that communicate with external entities but not in the application I am building (so far at least). Everything is self contained.

Different software development projects require different testing techniques and practices as they have different purposes and goals. Unit testing is great for testing bits of code, like testing specific methods or classes. Positive and Negative testing check to make sure possible inputs are valid or invalid respectively, it’s important to check for both cases as a working test on one end does not guarantee the opposite end will behave as expected. Exception testing helps ensure exception throws are properly implemented in order to save trouble down the line, providing insightful feedback without crashing the program. Boundary testing ensures proper functionality within a given range which can be important for a project with defined limits, ensuring your system will not allow invalid values or cause it to crash. It’s important to tailor your tests to your specific program, project, or system in order to get the most out of the testing process, ensuring all requirements are met.

**Mindset:**

While working on this project I was sure to be cautious of hastiness. Initially I had a few errors as well as failures within my tests. After examining them closely I found that the issues were caused by simple discrepancies in the code itself. Because I was working with so much at one time I was not as slow moving as I tend to be in order to ensure accuracy. Because of hastiness my initial tests had issues which did not promote my skills as a tester. Before submitting the final product I was sure to go over my code and tests with a fine tooth comb, finding the source of each error and failure which ended up being fairly simple bugs.

Limiting bias is important in any project, including software development and testing. While examining the requirements for the software there were some very specific criteria to consider, and some of these I had some personal issues with. The issues I had with the requirements were mainly about character limits for names. I thought to myself “If the limit is 10 characters that could definitely alienate some people, even if they are few in number.” If this was a project where I could discuss requirements with the client or even a go-between like a Project Manager, Product Owner, Scrum Master, etc I would have definitely made this suggestion. Regardless, it is important to be considerate of possible biases.

My last thoughts on this project is my commitment to quality. Work you present to others is a tangible representation of you as an employee, a developer, and even a person. While I had issues throughout the project's process, it was because I was learning. Before I presented the final product I did everything I could to provide a well tested and efficiently developed product. Laziness leads to a loss in time and money, so cutting corners is not an option for me personally. It is always better to do things right the first time, and a loss in money and trust because of hasty work or pure laziness would only hurt you in the long run.

Sources:

Atlassian. (n.d.). *What is Code Coverage? | Atlassian*. https://www.atlassian.com/continuous-delivery/software-testing/code-coverage