CS-320 Software Test Automation & QA

7-2 Project Two

Jacob Theisges

Southern New Hampshire University

December 19, 2021

It goes without saying that it is beyond critical that the testing approach aligns with software requirements. This is because, as we have learned, testing is a very important, if not the most important, part of the software development lifecycle. Testing is used to determine the functionality of software/code that is specified by the requirements given by the client. JUnit test cases allow developers to effectively test code against all requirements, in order to make sure the actual code and requirements align. After developing the classes Contact.java, ContactService.java, I developed test cases ContactTest.java, ContactServiceTest.java. I then the used the same approach for this week, creating Task.java, andTask\_Service.java, followed by the test cases TaskTest.java, and TaskServiceTest.java. By doing this, it allowed me to break down each requirement to be tested using JUnit testing; this approach ensured that each defined requirement was met. For example, for the “Contact” class, one of the requirements stated a first and last name could not exceed 10 characters in length. To adhere to this requirement, the “ContactTest” class Junit implemented an assertion, “Assertion.assertThrows” to test a given input. Because the requirement prevents an ID from exceeding 10 characters in length, a Junit in the “TaskTest”, “void testTaskIdTooLong()” was implemented to throw an error if an input was longer than 10 characters.

The overall quality of my JUnit tests, for the contact service and task service was surprisingly effective, as they were able to cover at least 95% of developed code. I say surprisingly because I am still new to coding, so I do not have a lot of confidence. Running JUnit testing ensured that all requirements were met. To be positive that my code was technically sound, I ensured that I followed the code annotation by properly naming all classes and methods. For example, I used the code @Test to declare the following method as a test method, whilst also using assertions to test the code against client requirements:

private String id, name, description;

private String tooLongId, tooLongName, tooLongDescription;

*@BeforeEach*

void setUp() {

id = "1234567890";

name = "This is Twenty Chars";

description = "The task object shall contain a required description.";

tooLongId = "111222333444555666777888999";

tooLongName = "This is way too long to be a task name";

tooLongDescription = "The task object shall have a required description String field that can be no longer than 50 characters. The description field shall not be null.";

}

*@Test*

void getTaskIdTest() {

Task task = new Task(id);

Assertions.*assertEquals*(id, task.getTaskId());

Also, to ensure efficiency, I tested the classes by passing examples into the code. This ensures that the written code is efficient, and functioning based off client requirements.

As we have learned throughout the duration of this course, testing is a vital part in the Software Development Lifecycle (SDLC). Moreover, there are numerous ways to test code; in this course, we complete our testing utilizing JUnit testing. For the first milestone, we created a package regarding contacts; the code allowed for the addition, deletion, and updating of a specific contact. JUnit testing techniques were used for this package to determine the functionality of the code. For the second milestone, we created a package regarding tasks, or more specifically, in-memory data structures. Likewise, to milestone one, Junit testing techniques was used to determine the functionality of the code. Lastly, for milestone three, we

created a package regarding appointments. Once again, JUnit testing was used to determine the functionality of all three packages to determine the functionality of the code.

Using the above stated technique, we were able to test specific parameters within the code to determine if a certain value was entered as “null”. Furthermore, we were able to test functions in the event an input went against the predetermined requirements. Junit testing techniques became very familiar during the course of the three milestones; however, there are other techniques that could have been utilized. Among these techniques is one called JBehave, which focuses on the behavioral side of testing. Although, I did not utilize this technique due to the fact that I am not familiar with it. In addition, JBehave requires a large amount of communication; I felt as if the milestones did not require this certain testing technique.

Because of my limited experience with coding, I engulfed a mindset of growth and learning. I knew going into this course that there would be challenges, so I told myself if I ran into an obstacle, I would find a way to go through it rather than around it. I employed caution by watching many tutorials prior to starting a class. By doing this, I was able to get a solid idea of what to expect and what to accomplish; I also performed numerous trial and error runs before finalizing a class/package. When it comes to complexity and interrelationships of the code, it is critical to note the importance. For example, upon first inspection of testing coverage, my coverage percentage was exceedingly low. However, I incorporated more tests that covered more of the code, which brought up my coverage percentage significantly. By adding more tests, increasing the coverage, this ensure my code was technically sound.

Prior to this reflection paper, I had not though about bias in a coding aspect. I made many assumptions through the duration of this course, and I am guessing that could be considered bias. However, I feel bias comes into play a lot more when a developer tests his or her own code. To highlight this, in my situation, had I only tested the function that verified the length of a first name to be no longer than 10 characters, I might not have realized that the ID test errored out due to a missing line of code, or vice versa. Incorporating tests that verify both valid inputs and invalid inputs, I feel limits bias for developers.

I firmly believe in quality in everything I set to accomplish. When it comes to programming, quality is very critical. After all, the majority of everything around us is a program a development team released. Programmers, in a sense, are public service workers. The people or businesses they develop for, have day to day contact with the public. Cutting corners invites an ample number of problems that can be detriment to everyone involved. Having solid work ethics ensures quality products are delivered. All of this falls under the umbrella of emitting self-discipline and showing integrity.