John Colon

CS 305

Project 2

Project 2 Reflection

When diving into the unit testing, requirements were already set up for me to understand what the code needs to do. Example of this is when the company needed the lastName string to not be larger than 10 characters and not be null. This meant that I needed to make sure that the unit test, tested for anything that was greater than 10 or equal to null. I would do this for every string I needed. An example of the requirement that needs to be passed in my code using the last name input is:

“protected void updateLastName(String lastName) {

if (lastName == null || lastName.length() > CONTACT\_LNAME\_LENGTH) {

throw new IllegalArgumentException("Last name cannot be empty or have more than " + CONTACT\_LNAME\_LENGTH + " characters");

} else {

this.lastName = lastName;

}

}.”

This used the byte I set up within CONTACT\_LNAME\_LENGTH which equalled 10, to make sure that the input was not higher than 10, and the null made sure that it did not equal null.

These requirements need to be tested to make sure that the code has little to no errors, by testing different modifications of information like adding, deleting and/or updating the imputed information. This is tested in my code through the use of JUnit. An example of this test is:

“@Test

void contactIdAndFullNameConstructorTest() {

Contact contact = new Contact(contactId, firstNameTest, lastNameTest);

assertAll("constructor three",

() -> assertEquals(contactId, contact.getContactId()),

() -> assertEquals(firstNameTest, contact.getFirstName()),

() -> assertEquals(lastNameTest, contact.getLastName()),

() -> assertNotNull(contact.getPhoneNumber()),

() -> assertNotNull(contact.getAddress()));

}.”

Now with my experience with JUnit testing, I had a bit of trouble figuring out how to execute the code. The code needs to be practical and effective while also being simple enough to be able to know what each part is testing. With the code I made I did have to do a few different builds to understand how to make the first one, but with the final one, I knew it was done in a way that was clean. One thing that is needed that I forgot to do at times is adding comments to your code. This allows the developer and anyone else to know what is going on.

One thing I did to make sure that was more effective in my code was combine the requirement test of the length and null. This shortened the check even though it added more to the error sentence that notified the user of an error for null or exceeding length:

“//taskId set

private void checkTaskId(String taskId) {

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

throw new IllegalArgumentException("Task Id is invalid. It must be shorter than 10 characters and not be empty.");

} else {

this.taskId = taskId;

}

}.”

This makes sure to test all requirements, while making sure that different possible issues are found. In my taskTest.java, I made a temp name to test the code to ensure that the requirements are met, and makes sure to test that the task name is not too long:

“@BeforeEach

void setUp() { …  
 name = "This is twenty chars"; …

tooLongName = "The task name cannot be this long."; …

@Test

void getNameTest() {

Task task = new Task(id, name);

Assertions.assertEquals(name, task.getName());

} …

@Test

void setTooLongNameTest() {

Task task = new Task();

Assertions.assertThrows(IllegalArgumentException.class,

() -> task.setName(tooLongName));

}.”

This tests to make sure that the code is following all the requirements making sure that it will still keep the name that is equal to or less than the required length.

REFLECTION:

Each time I code I would do functional testing which is manual testing and unit testing. For functional testing, when executing manual testing, I would have someone test a section of my code by using the system. This is usually after I have done my own unit testing, but I would allow an outsider to test the code and give them an idea of how the code is supposed to be used. I would have them look through the code and see the code in their eyes. I do this due to me sometimes getting a bit blind to my own work. Manual testing is good to get fresh eyes in the way I use it.

For unit testing, it is when you are testing specific sections of the code that was worked on. I usually do this after I make a block of code, to make sure that the block has no error. It allows me to fix it in the moment rather than when I finish.

I could have used Decision table testing that would allow for testing of defined conditions that functions would operate, while defining the functions as well. This would test combinations of conditions and actions to show the different combinations that a selection could have.

MINDSET:

When working on the code, I had to adopt a mindset of problem solving and putting myself in the shoes of a user. This allowed me to understand what was needed to make sure that if a user placed an incorrect input, they would know why and how to fix it. As Blake states, “Great testers are able to imagine themselves in the place of the user, to predict what they might do, how they might be confused, or why they might get frustrated,” (Norrish, Blake). This is something I adopted when thinking about my testing. How would the user be affected by a miss test or not knowing what they did wrong.

Trying to be less biased to your own code can be difficult. When testing my code, I would first walk away for a bit to then come back with the thought of fixing any mistake. This is actually something I do with my experience of being a writer. When I was in script writing class, my professor would let me know to take a set away and come back after a day or an hour, however much time you need. This will clear your mind and detach you from your work to gutt it like it needs to be. My professor’s advice helps me everytime when I need to deeply fix any type of creative project that feels like I pour my heart out like coding.

When working on coding, I have to make sure I consider consistency, efficiency and not cutting corners. If I were to cut corners it would show up in my coding with either errors or missing work. I need to make sure to take time with my code to develop it with efficient use of time. If I were to skip testing sections or skip rereading my code, it will cause bad habits and cause more issues for me in the future. A skill is honed by working on it one step at a time, not skipping out on sections that need to be worked on, no matter if it is difficult.

REFERENCES:

GeeksforGeeks. (2023, December 6). *Software testing techniques*.

<https://www.geeksforgeeks.org/software-testing-techniques/>

Norrish, B. (2021, October 15). *How to think like a tester*. Medium.

<https://medium.com/@blakenorrish/how-to-think-like-a-tester-7a174ff6aeaf>

*Software testing techniques: Explained with examples*. BrowserStack. (2023, May 25).

<https://www.browserstack.com/guide/software-testing-techniques>