CS 320 Module Seven Project Two: Summary and Reflections Report

Kelly Illescas

June 19, 2022

When I first began working on the JUnit tests, I found that I had a significant amount of trouble getting my code to align with the software requirements. I thought doing this would be very simple, but realized that it was more complicated than I expected. I created the Contact class and had no trouble with it whatsoever. Equally easy was the ContactTest class for the JUnit test. I feel that these were easy because I had the video tutorial to help me. However, when working on the ContactService class, I had trouble initially with the add and remove parts. Again, thinking it would be easy, I was not worried about it; however, it took a lot of manipulation to finally get it to work. The ContactServiceTest class was the most difficult for me. The test kept failing, and no matter how much I worked on it I couldn’t seem to get it to resolve itself. I then turned to YouTube to try and figure it out. I came across a great video that outlined exactly the trouble I was having. This was with the error “cannot be resolved to a type.” I noticed that the person who made the video had imported both the Contact and ContactServices classes. I had only imported the ContactService class. Once I did that, all the tests passed. The same thing had happened with the Task project. The classes were very easy to code. I simply copied and pasted what I had coded from the Contact project, then edited the coding by changing the contact information to task information. I once again had to import both the Task and TaskServices classes into my TaskServiceTest JUnit class, and when I did that, all the tests passed. I know that my JUnit tests were quality tests, as all tests passed successfully.

I ensured that my code was technically sound because I compared it to the video tutorial provided by SNHU. By making sure the code written aligned with what was provided in both the video and in the PDF that was provided, I knew that when I tested it, I would be successful. In particular, I used lines of code to validate that the Name could not be longer than 20 characters and could not be null. The code looked like this:

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

throw new IllegalArgumentException("Invalid Name");

}

I used similar conditions for the ID, Phone Number, Address, and Description in both projects. They were all technically sound. both the ContactService and TaskService classes, I used the add() and remove() features to add and remove contacts. These were very useful because I did not need to write very many lines of code due to this great built-in feature. The code looked like this:

I ensured that my code was efficient because I tested it with several different parameters. By adding, deleting, and updating different aspects of both the Contact and Task projects, I was able to get all parts of the code to test successfully. In both the ContactService and TaskService classes, I used the add() and remove() features to add and remove contacts. These were very efficient because I did not need to write very many lines of code due to this great built-in feature. The code looked like this:

//adding contact

if (!existingTask) {

tasks.add(task);

System.out.println("New task added successfully.");

return true;

}

else {

System.out.println("Task already exists.");

return false;

}

}

//Delete tasks by ID

public boolean remove(String TaskID) {

for (Task t : tasks) {

if (t.getTaskID().equals(TaskID)) {

tasks.remove(t);

System.out.println("Task deleted successfully.");

return true;

}

}

System.out.println("Cannot delete. Task does not exist in system.");

return false;

}

For each of the milestones, I employed a functional testing technique known as JUnit testing. JUnit testing is a regression testing technique that focuses on testing small amounts, or units of code, to be sure that changes to it will not adversely affect the program as a whole. JUnit testing is a subset of unit testing, which is itself a form of white box testing. When unit testing, “test cases are based on internal structure” (Chacon, 2018). When writing my JUnit tests, I used tools such as assertEquals to simplify my technique, and add() and remove() to make my code more efficient.

For these milestones, I did not use integration testing, system testing, or acceptance testing. Integration testing allows a tester to test two separate systems as one. System testing occurs when a tester evaluates the program as a whole, as opposed to unit testing where only small units are tested at one time. Examples of system testing include End to End testing and Black Box testing. Acceptance testing involves real-time testing of a program. This occurs when the buyer or customer tests the program. Examples of acceptance testing are alpha testing (the company tests the product) and beta testing (the customer tests the product).

The practical implications for JUnit testing include simplicity of understanding units and the universal ability to collaborate with engineers. Because JUnit testing is done on such a precise level, the results of testing and its subsequent debugging are more easily understood. This way, the code can be debugged quickly and efficiently. In addition, since JUnit testing is fairly universally coded and can be run on almost all IDEs, it comes with the capability of being shared amongst developers, who can collaborate and provide ideas, changes, and quick fixes to make the code even better.

I had to really focus my mindset while I was working on this project. First, I had to employ caution while creating the tests because I had never tried something like this before.

JUnit testing was completely foreign to me, so I had to be extra careful when typing out certain parts of the code. I had to watch many YouTube videos and look at the information from our Resources section to garner enough prior knowledge to be able to decide how to code these tests. I was cautious in that I ran the tests with as little coding as possible at first, then slowly adding more as I went along. When running the tests initially, I had errors and had to submit the milestone due to time constraints. After going back in and reworking the code many times, however, being cautious as to exactly what I was typing (spelling, punctuation, and spacing), I found that I was more easily able to solve the problems with my code and fix the errors. Finally, I used caution by trying to replicate as much of the code as possible in order for any package to be easily read by anyone who may need to access it. Here is an example of repetitive code:

ContactServiceTest.java

//checking the add new contact feature

@Test

public void testAddNew() {

ContactService cs = new ContactService();

Contact c1 = new Contact("000", "Kelly", "Illescas", "6031234567", "4 Ellison Drive");

assertEquals(true, cs.add(c1));

taskServiceTest.java

//checking the add new task feature

@Test

public void testAddNew() {

taskService ts = new taskService();

task t1 = new task("000", "Kelly Illescas", "This is a description of the task");

assertEquals(true, ts.add(t1));

To limit bias in testing, it is important for a tester to be generally unaware of who created the code a person is testing. While this is not always possible, it is an important step. Too often, it has been noted that there are not enough women or people of color in the field of software engineering. There have been numerous studies dedicated to identifying the why of this. In one particular case, programmer Jason Gorman noted that he conducts “blind programmer coding evaluations” (Johnson, 2014). In the article, Phil Johnson (2014) notes that the blind evaluations were conducted by simply viewing code; there was no prior meeting of the interviewee, speaking with that person over the phone, or even reviewing that person’s CV. This way, the evaluator would simply choose the best candidate by way of that person’s work. I admit that I did not actively try to limit bias when reviewing my code. It is an incredibly important step in the world of software engineering, however, as it can open doors where they were firmly shut before.

Finally, I had to focus a lot on being disciplined when creating my code and subsequent JUnit tests for this course. I had to begin each of my milestones early in order to be sure I would be finished on time and in case there were snags, which actually did happen for me. I had the unfortunate experience of losing all of my previously created work on Eclipse. This happened after I had finished my second milestone and saved it, but had not yet submitted it. After much investigation and conversation with SNHU’s IT department, I ended up finding out that files are routinely deleted from our server. I had not saved any of my work on my local computer, so I had to begin again. In this case, discipline is so important. The snag I encountered was a big one for me, as I had to start hours of work over from scratch. In the future, I will be more disciplined by being sure to save my work both on the virtual drive and on my local machine. In the world of programming, a program lost could potentially contain thousands of lines of code or more. If a program like that is not saved frequently, it could cost considerable time, effort, and money that some companies might not be able to recover. Discipline in that regard is absolutely essential.

References

Chacon, D. (2018, Aug 2). *JUnit tutorial: Setting up, writing, and running Java unit tests*.

Parasoft. Retrieved from <https://www.parasoft.com/blog/junit-tutorial-setting-up->writing-and-running-java-unit-tests/

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2015). *Software testing:*

*An ISTQB-BCS certified tester foundation guide* (3rd ed.). BCS Learning & Development

Ltd. Retrieved from <https://app.knovel.com/hotlink/toc/id:kpSTAIST01/software-testing-an-istqb/software-testing-an-istqb>

Johnson, P. (2014, Mar. 19). *How blind coding evaluations could reduce interviewer bias*.

ComputerWorld. Retrieved from <https://www.computerworld.com/article/2701238/how-blind-coding-evaluations-could-reduce-interviewer-bias.html>

Software Testing Help. (2022, May 5). *Types of software testing: Different testing types with*

*details*. Retrieved from <https://www.softwaretestinghelp.com/types-of-software-testing/#Functional_Testing>

Southern New Hampshire University. (n.d.). *Tutorial using JUnit*.

<https://learn.snhu.edu/content/enforced/1069812-CS-320-T5619-OL-TRAD-UG.22EW5/course_documents/CS%20320%20Module%20Four%20Tutorial%20Using%20JUnit.pdf?_&d2lSessionVal=5A5o3l9cb81sW5RgK9fgYUwG3&ou=1069812>