Breunna Bingham

CS-320 Software Test Automation & QA

Southern New Hampshire University

December 6, 2022

Project Two

Describe your unit testing approach for each of the three features.

* To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.
  + For each of features, I made sure to read each rubric and all the rubrics asked for a specific character count to limit each input by. I specified the character count to check for in the code, when running my JUnit tests for each assignment. I also checked if fields were null and tested the fields by placing a null as an input in the Junit tests. Some of the features asked that certain fields be updated while others were not to be updateable. I created getters and setters for the updatable fields and created Junit tests to specifically test the updatable fields. Here is an example of the code I wrote for the updatable fields in TaskService.

public boolean updateTask(String taskId, String name, String description)

{

for(Task task : taskList)

{

if(task.getTaskId().equals(taskId))

{

task.setName(name);

System.***out***.println("The name field has been updated.");

task.setDescription(description);

System.***out***.println("The description field has been updated.");

return true;

}

}

System.***out***.println("The task with the unique TaskID was not found");

return false;

* Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?
  + When writing JUnit tests, I made sure that I was covering as high of a percentage as possible for each of the files that I was testing. When I ran it as a coverage JUnit test, specific lines of code would be green if successfully tested, whereas others were red. With my personal experience I noticed that all my “return false” lines of code were not being tested, so I had to ass more lines of code to my testing file and assert them as false to get those lines of code to turn green and be successfully tested which also elevated my overall testing percentage. Throughout all the assignments I always strived to get at least 85 percent of testing coverage to each of the files I tested. This ensured that I was thoroughly testing all aspects that I needed to be tested to meet the rubric requirements.
* How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

public void testUpdateTask()

{

Task firstTask = new Task("12345", "Breunna", "Update address book within app");

Task secondTask = new Task("23456", "Amber", "Test address book funtionality");

Task thirdTask = new Task("98765", "Janelle", "Create functionality for favorites in address book");

TaskService test = new TaskService();

*assertEquals*(true, test.addTask(firstTask));

*assertEquals*(true, test.addTask(secondTask));

*assertEquals*(true, test.addTask(thirdTask));

//updating name for First Contact

*assertEquals*(true, test.updateTask("12345", "Shyann", "Update address book within app"));

*assertEquals*(firstTask.getName(), "Shyann");

//updating name and description for Second Contact

*assertEquals*(true, test.updateTask("23456", "Samantha", "Test functionality to add contact"));

*assertEquals*(secondTask.getName(), "Samantha");

*assertEquals*(secondTask.getDescription(), "Test functionality to add contact");

//updating description for Third Contact

*assertEquals*(true, test.updateTask("98765", "Janelle", "Create new functionality for favorites"));

*assertEquals*(thirdTask.getDescription(), "Create new functionality for favorites");

//Testing the false assertion here with null passed in

*assertEquals*(false, test.updateTask(null, null, null));

* I’m going to use the example from my TaskService assignment. As you can see above, I ensured that my code was technically sound by making sure that my code was tested for all functionality that was written. An easy way of checking for this is to run the JUnit test as a coverage assessment. For example, as you can see above, I wanted to catch the instances where the user may enter nothing or null if you will, for each of the fields, in which that is not to be accepted as input. So, I made sure to test for all areas, such as updating the task’s description and name. I also tested for the false assertion if the user entered nothing for the fields. Again, it was confirmed that I was testing every line of code when I ran the JUnit test that included the coverage assessment. If the lines for code came back red, then it confirmed to me that that specific line of code was not being tested, so I had to make some changes to the JUnit test to make sure that all the lines in my files appeared green, which indicated proper testing has been successful.
* How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

public boolean updateTask(String taskId, String name, String description)

{

for(Task task : taskList)

{

if(task.getTaskId().equals(taskId))

{

task.setName(name);

System.***out***.println("The name field has been updated.");

task.setDescription(description);

System.***out***.println("The description field has been updated.");

return true;

}

}

System.***out***.println("The task with the unique TaskID was not found");

return false;

* + Above is another example from TaskService assignment, where I created print statements to let myself know when a task was updated, and which areas were updated. I also included a print statement that told me that the unique ID that was entered was not found, letting me know that nothing was updated. By creating these print statements, it was easy to follow along in the console when performing the JUnit tests. Initially I did not do this for my first assignment ContactService, but I went back and added this in for Project one, because with the other two assignments, It was easy to follow along the JUnit tests and visualized the areas they made contact with successfully. If perhaps I got an error, the console with the print messages really helped see where the error was and how I needed to fix it. I did this for all my tests so that I can confirm that my code was running exactly as expected from each of the rubrics.

1. **Reflection**
   1. Testing Techniques
      * What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.
        + By far, I think the technique that I used most often was Dynamic testing. Once my files were finished being written, and then I wrote a testing file for that file, I was then able to successfully test. I was constantly running my JUnit tests to see the coverage percentage and I made small changes over and over to get my percentage to raise on each file. Some testing techniques that I used a lot again, were the print statements to allow me to visualize where the code was successfully being tested and passing. I also really enjoyed running the coverage JUnit tests to pinpoint exactly which lines were being checked and which were not by the color coordination feature. If I noticed that anything was red, I went back to my JUnit tests and added more lines of code that would specifically check the lines of code that were not covered. In most cases these were the assert false lines of code that I needed to add to the testing files.
      * What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
        + As much as I am a Static Tester in every aspect of coding, this class made me more into a Dynamic tester. The reason for that is with Static testing you test as you write the code. It helps to eliminate bugs along the way, and it creates a more efficient work environment as you progress in the project. The problem with JUnit Tests and static testing is that it’s hard to perform testing on these files until they are completely written and just needs subtle changes. This was at least my personal experience with this class. I personally thought static testing with JUnit tests didn’t make a lot of sense, when Dynamic Testing is more effective.
      * For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
        + So, as I explained a bit already, Dynamic testing is great when you are working with Junit tests that you are trying to write out tests to test each line of your file successfully. Static testing is great in almost every other instance. I personally love static testing more than Dynamic because I get to check my work, and the thoughts I have about changes frequently. I can easly catch where I made mistakes because I find them right after I make them. Whereas with Dynamic testing, there might be quite a few mistakes, which in the end causes more confusion when you have more lines of code to sort through to find your issue.
   2. Mindset
      * Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.
        + This class was a bit different for me. It put us in a different seat as a developer and made us think differently about a program overall. I think what stood out to me the most was that as I was writing the files that were going to be tested, I was thinking at the same time “What is the easiest way to write a JUnit test for this?”. And that type of thinking helped me remain cautious with how I wrote the code. I didn’t want to confuse myself with redundant code that was unnecessary and therefore more lines of code to test. I used a lot of the same techniques and layout in each of assignments to keep things familiar to me and making the most sense.
      * Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.
        + I wouldn’t say that I eliminated bias with my first assignment ContactService, at least not initially, because this was my first assignment, I was still trying to understand how everything worked and how JUnit testing was performed. I knew how to write code, and for the most part, the rubric guidelines were simple, so I assumed everything I was writing would be perfect on the first try. Upon working on the other two assignments, the more and more I worked with JUnit tests, the more and more I became confident in my abilities to write out JUnit tests and get a high percentage get a high percentage of coverage. The great thing about Junit tests is that they help to eliminate that bias associated with testing your own code. Even the best software developers make mistakes, and JUnit tests can help find them without the use of peer reviews.
      * Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.
        + Before this class, I had no idea what a JUnit test was. I come form a Nursing background, so these classes I am taking now, are all new to me. Throughout my educational career in this field, I have found myself cutting some corners when writing code, where it as only come back to haunt me. As I code more and more, and I can learn from my mistakes, it helps to identify the smarter decisions for the future having made those mistakes in the past. Now that I know how to use JUnit tests, I can write up my own tests to make sure I am testing my code properly and thoroughly. In the workforce, time is money. There are deadlines to meet and clients expecting results. When developers cut corners and are not through in their testing abilities, this can cause major setbacks on the project, unhappy clients, and a loss of money for the company. It is imperative that developers test thoroughly to avoid costing companies more money and potentially loss of customers.

References

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.