**4-2 Journal: Unit Testing Approach and Writing JUnit Tests**

Christine R. Emerson

Department of Computer Science, Southern New Hampshire University

CS-255: System Analysis and Design

Dr. Toledo Lopez

March 6, 2023

* **To what extent was your testing approach aligned to the software requirements? Support your claims with specific evidence.**

The testing phase is essential in detecting defects/bugs, ensuring software requirements are met before delivering the software to the client and guaranteeing the quality of the software. In addition, it makes the software more reliable and easier to use. Software that has been thoroughly tested ensures a high level of performance, reliability, and efficiency. Testing with JUnit ensures developers' code meets the client's requirements. I divided the requirements into more manageable and smaller sections to create my test cases. For example, in Task.java, I started by ensuring that the task Id, name, and descriptions were not null, had correct lengths, and were updatable (if applicable). The process begins by throwing illegal arguments (error messages) when the length is incorrect or if the argument is null. Next, setting the variables in Task.Java enables them to be updated because the only one that should not be updated is the task id which was not set. For example:

Graphical user interface, text, email

Description automatically generated

In TaskTest.java is where I used jUnit tests to ensure those requirements were being properly implemented, for example, with setting the task name:

Graphical user interface, text, email

Description automatically generated

* **Defend the overall quality of your JUnit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**

It is always necessary to analyze the entire code for all test cases. As a result, we have definitive and statistical proof that any code that can be tested has been tested. In general, 80% or higher test coverage is considered reasonable. When running the JUnit test on the ContactService and TastService code, the Junit and Coverage tests were conducted to measure how much of the code ran during the unit test. As the completed test indicates for the ContactTest.java class file, the JUnit test was completed quickly (0.011 s), and the TaskTest.java class file (0.025 s) with both files ran with no error or failures. As shown below:

Graphical user interface, text, application, email

Description automatically generated

Indicating that the code is intact, efficient, and error-free; however, these JUnit tests do not cover incorrect implementations. Looking at my code coverage below for both the contact service and task service classes:

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated with low confidence

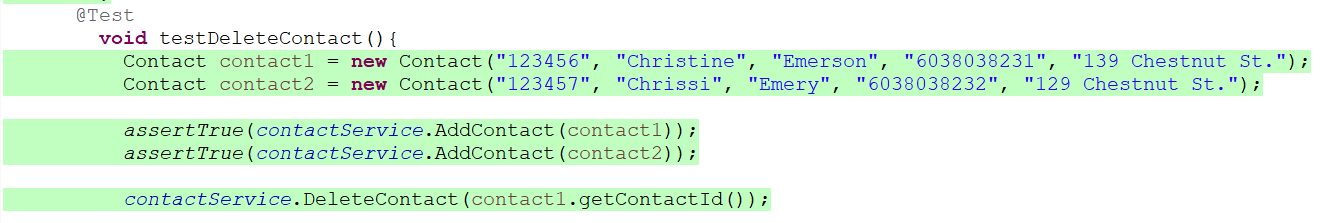
You can see that both classes have over 80% coverage, indicating that the percentage of tested code was effective. I created effective testing by checking that my work aligned with the requirements. Each part of the test case and each test case were covered within the requirements given. For example, in ContactService.java, once I finished one method, I would go to ContactServiceTest and write the test for that method. As shown:

***ContactService.java***

Graphical user interface, application, Word

Description automatically generated

***ContactServiceTest.java***

******

Then once all was done, I went through the contactService to ensure all requirements were met, such as the contact service was able to add contacts with a unique ID, delete contacts per contact ID and update contact fields per contact ID (firstName, lastName, phoneNum, and address updatable). Finally, I did the same for the taskService class; the task service was able to add tasks with a unique ID, delete tasks per task ID, and update task fields per task ID (name and description updateable). As a result, this ensured that the requirements were met and the JUnit tests were effective.

* **How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

I ensured my code was technically sound by making it syntactically accurate, logically concise, and modular. For example, below, the code searches the dictionary to get a contact id instead of going through every id individually and uses Boolean to minimize unnecessary for loops which helps keep code minimal and organized, as shown:Graphical user interface, text, application

Description automatically generated

I kept the code simple as possible and met each requirement without error. The code has been tested and functions as expected. For example, in the task class, I used an if statement to iterate over each field to ensure they were not null and over the required length and throw an exception if the field does not meet those requirements: A screenshot of a computer

Description automatically generated with medium confidence

Then each value is assigned, and the getter methods are established to access each variable from other classes. As well as setter methods so certain variables can be updated.

* **How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

To ensure my code was efficient, I removed any unnecessary code or code that goes to redundant processing. I ensured the best speed or run time for completing the algorithm, as shown:

Graphical user interface, text, application

Description automatically generated

I made use of reusable components wherever possible. I made use of error and exception handling at all layers of software. The exception handling of code can be traced using JUnit. The code can be tested to see if it throws a desired exception. @Test annotation is used along with the expected parameter, as shown in **TaskTest.java**:Graphical user interface, text, application

Description automatically generated

In addition to developing programming code compatible with the design logic and flow, I also followed appropriate coding practices. For example, I used the best keywords, data types and variables, and other available programming concepts to implement the algorithm.