# **CS 320 Module 4 Journal**

Andrew Emilio DiStefano

SNHU

Professor Omar Toledo Lopez

June 1, 2024

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

My testing approach for the ContactService milestone aligned with all of the software requirements as specified in the prompt. My code assured that the contact ID string, the first name string, the last name string, the phone number string, and the address string of each contactall contained the appropriate numbers of strings as specified in the requirements by using ‘if’ statements to throw an error for strings which did not meet these requirements. I also included arguments representing null values for each of those ‘if’ statements so that an error would also be thrown if any of those values were left blank or otherwise not set. I then added functions which would allow the user to add contacts by unique ID, delete contacts by ID, and update the first name, last name, phone number and address of each contact. The ID was left unable to be updated as specified in the requirements.  
  
 My testing approach for the ContactServicemilestone aligned with all of the software requirements as specified in the prompt. My code assured that the task ID string, the task name string, and the task description string of each task all contained the appropriate numbers of strings as specified in the requirements by using ‘if’ statements to throw an error for strings which did not meet these requirements. I also included arguments representing null values for each of those ‘if’ statements so that an error would also be thrown if any of those values were left blank or otherwise not set. I then added functions which would allow the user to add tasks by unique ID, delete tasks by ID, and update the name and description of each task. The ID was left unable to be updated as specified in the requirements.

* **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?**

“As a general rule, a test coverage rate of 80% or above is considered reasonable” (Garcia, 2017, p. 284). My JUnit tests for the contact service had a coverage percentage of 90% over my contact service code. My JUnit tests for the task service also had a coverage percentage of 90% over my task service code. I achieved this level of coverage by using the same methods which I had used to provide high coverage over my contact service code. At first I had some difficulty providing enough coverage in my test files, but after performing some additional research I made sure to add relevant test cases to my test files. Having carefully gone through my code for each milestone line-by-line and having added a test for each possible error which could be thrown, and having then run my code with coverage in IntelliJ IDEA, I can rest assured that my JUnit tests were effective on the basis of coverage percentage.

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

I used several techniques to ensure that my ContactService and TaskService code was technically sound. These included the performing of tests where a new service was created (which should start out empty) and then checking which contact (or which task for the task service) could be found at index ‘0’. I used this method to test my method for removing a contact as follows:   
  
void testRemoveContact() {

ContactService service = new ContactService();

service.addContact("550", "Elmer", "Fudd", "5904454545", "21 Dreary Lane");

service.addContact("202", "Muffin", "Man", "8006833467", "18 Dreary Lane");

service.removeContact("550");

assertEquals("202", service.contacts.get(0).getContactID());

}

Not only does this test assure that the contact has been removed by checking that the second contact added is found at the first index of the array list, but it also assures that the new service is created empty and not with other unnecessary elements added upon creation of a service. I used a similar method in my task service code as follows:   
  
void testRemoveTask() {

TaskService service = new TaskService();

service.createTask("550", "Default name", "Default description.");

service.addTask("550");

service.removeTask("550");

assertEquals(service.taskList.get(0).getTaskID(), "1234567");

}

Again, this test assures the effectiveness of the ‘removeTask()’ function while also making sure that the service is not created with other elements involuntarily included upon creation.

Another way in which I ensured that my ContactService and TaskService code was technically sound was the adding of print statements to write out the arrays contained in my array lists (where the contacts in ContactService and where the tasks in TaskService were stored) after each function so that I could visually see the contents of each arraylist and assure that objects were being created, deleted, and modified correctly by my code. These were written into my Contact, ContactService, Task, and TaskService files rather than being written into my tests. These were not included in my submitted code since these print statements were not part of the requirements, but they did help me to ensure that my code was technically sound.

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

One way in which I ensured that my code was efficient was the placing of multiple arguments into my ‘if’ statements responsible for ensuring that strings met the requirements of the ContactService and TaskService milestones. For example, the first line of my ‘if’ statement confirming the specifications for the contact ID length string was as follows:

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

This use of the logical ‘or’ operator (‘||’) allowed me to avoid having to write separate ‘if’ statements for each of those arguments. The above example was copied from my ContactService code but I used the same technique in my TaskService code as well, for example to confirm the specifications for the task description string as follows:

if (taskDescription == null || taskDescription.length() > 50) {  
  
Since at first I had incorrectly used the ‘.length()’ method included in the Java ‘String’ class, the code was not effectively ensuring that the task description string was not null. The following test code in the ‘testTaskNull()’ test in my ‘TaskTest.java’ file alerted me of that fact:

assertThrows(IllegalArgumentException.class, () -> {

new Task("8675309", "Default Name", null);

});

**Sources:**

García, B. (2017). *Mastering software testing with JUnit 5*. Packt Publishing.