Victor Feight

3/26/2021

CS-320

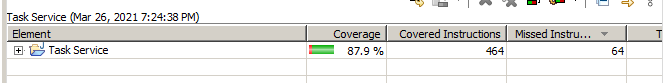
* To what extent was your testing approach aligned to the software requirements? Support your claims with specific evidence.

I have tried to create a Test Driven Development environment in which my test data and assertions were derived directly from the Software requirements for both Task Service and Contact Service components. Specifically, for the contact Service, I employed the technique of Equivalence Partitioning on the edge cases of the required phone number length; I did this by breaking up partitions into valid and invalid cases, 9, 10, 11 digits (with 10 being the correct amount). I ensured IllegalArgumentExceptions were thrown at all of the edge cases; this also implements boundary value analysis. For each requirement, I added a test case for both programs. Examples include adding test cases for “id too short”, “first name too short”, “last name too short”, in addition to checking for null values in all cases. Thus, I would say that my testing approach was directly aligned to the software requirements.

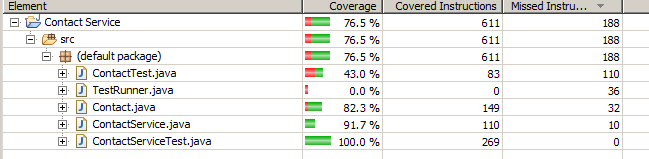
* Defend the overall quality of your JUnit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?

In addition to going through a checklist of all requirements, and creating a JUnit test per each requirement, I also knew my JUnit tests provided an effective coverage percentage by checking within Eclipse directly. By right clicking “Task Service” Project within Eclipse and clicking “Test Coverage As” 🡪 JUnit test, I was able to view an overview of my coverage percentage.

As an example, my coverage for Task Service project overall was 87.9%.



My coverage for Contact Service was slightly less, at 76.5%, though still pretty high. This is because I focused purely on testing the requirements for these functions:



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

To ensure code was technically sound, I ensured that any object fields that were to be instantiated were first verified before instantiation. This involved adding verification code preventing null values, lengthy fields, or short fields from being added to the object upon creation, instead throwing an Exception. I made sure to Assert these Exceptions were being thrown in my test cases, leading to more technically sound code:

private static void verifyID(String id) {

if(id == null || id.length() < 10) {

throw new IllegalArgumentException("ID must be > 10 chars, can't be null");

}

// id too short

Assertions.assertThrows(IllegalArgumentException.class, () -> {

Contact contact = new Contact("001", "Johnny H Cartoon", "Bravo Xaviar Network III", "9101234545", "123 Jay Way");

});

In addition, by testing at edge cases (using boundary value analysis), and testing each individual function, I was about to ensure the overall code was more technically sound and data flow was working appropriately.

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

I ensured my code was efficient and secure by using an Object Oriented approach, utilizing encapsulation for private fields, allowing them minimal privilege through access functions only as required. I modularized each function to enable easy testing of the codebase, for example, breaking addTask functionality into its own function allows easier refactoring later on. I tried to eliminate redundant code or logic and ensured any declared variable was used effectively. I also made most functions into Booleans to more easily give True or False values upon testing these functions. Lastly, I added comments describing the general purpose of my functions, variables, or tests enabling easier reading and rewriting of the codebase.