Project 2

Sean J Born

CS 320

October 20, 2024

**Summary:**

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

I used unit testing to test the code against the specific requirements lined out by the clients. One thing that I did to make sure that my testing aligned with the requirements is to chink each of the requirements into smaller sections such as updates, changes, additions, and then deletions. Chunking the requirements helped me make sure all requirements were met.

* **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?**

I believe that the overall quality of my Junit tests were effective in testing a majority of my code. I created tests that tested each of the requirements laid out by the client. For example one of the tests for each of the assignments was to test to make sure each parameter was within a certain number of characters. The test that I created tested to make sure that each parameter stayed within those number constraints.

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

To make sure that my code was technically sound I used code annotation by specifically naming the classes and methods used. Here is some lines of code from TaskTest.java:

**class** TaskTest {

/\*

\* The first 3 tests make sure the fields follows length constraints

\* The last 2 tests ensure that each field is not null.

\* TaskID is not tested for being not null.

\*/

@Test

@DisplayName("Task ID cannot have more than 10 characters")

**void** testTaskIDWithMoreThanTenCharacters() {

Task task = **new** Task("Name", "Description");

**if** (task.getTaskID().length() > 10) {

fail("Task ID has more than 10 characters.");

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

I made sure the code was efficient by making my code easy to update by using classes. For example, all length requirements are within a section that is labeled setters in the task.java file. This will help find these requirements to be changed in the future if need see code below:

// SETTERS

**public** **void** setTaskName(String taskName) {

**if** (taskName == **null** || taskName.isEmpty()) {

**this**.taskName = "NULL";

} **else** **if** (taskName.length() > 20) {

**this**.taskName = taskName.substring(0, 20);

} **else** {

**this**.taskName = taskName;

}

}

**public** **void** setTaskDesc(String taskDesc) {

**if** (taskDesc == **null** || taskDesc.isEmpty()) {

**this**.taskDesc = "NULL";

} **else** **if** (taskDesc.length() > 50) {

**this**.taskDesc = taskDesc.substring(0, 50);

} **else** {

**this**.taskDesc = taskDesc;

}

}

}

**Reflection**

Milestone one had me creating classes that allowed me to add and make changes to a contact list. Milestone two did the same but with specific tasks and their requirements. Lastly module three used in-memory data structures to support appointment services. In milestones one, two, and three the code was tested using Junit testing. Junit testing allowed me to test each of the requirements for each of the classes. This made sure that the code met the specified requirements as well as functioned properly. Another testing technique that I did not use is JBehave. This type of testing focuses on the behavior of the software being developed. This type of testing is more suited for much larger projects that need lots of communication. Each of these techniques are used to ensure programs are running correctly as well as make sure that all requirements outlined are present in the final product.

While working on the project I had to definitely be flexible and adjust to the new way of writing code since I have never written test before this course. I do not believe that I employed very much caution when writing my code. I say this because when implementing the test I checked to see if the tests I created passed but paid little attention if they failed. It is important to appreciate the complexity and interrelationships because a simple line of code can do so much and testing that can take several tests to make sure it is all functioning properly. It is important to make you are not being biased when reviewing the code because you want to make sure that the review is accurate. Making sure that the code is being reviewed without bias prevents errors from getting covered up and overlooked in testing. Being disciplined in the commitment to the quality as a software engineer is important for a number of reasons. The first is that it builds a good reputation of good quality product which can lead to more and better business as well as repeat clients. Another is that it helps save time and money during development because things are not having to be redone as often as it would if quality wasn’t a focus.