Kristen Hawkins

CS320 Week 7

Journal

**Summary:**

The approach that I took to align all the software requirements was using Junit testing. I did apply this to all the features needed for this project. While using the Junit testing feature, I found it to be effective in the project after noticing the percentage of the test that took place. The test was in the high range of performance, but I did come across one section of my test that had some lower counts. I believe it to be a lack of use the code had. Other portions of my code were effective according to the percentage rate.

Using the Junit testing for this project was a useful testing experience in this project. I knew that the setter and getters in this project were technically sound when they came no errors and past all testing done at the end.

**protected** **void** setFirstName(String firstName) {

**if**(firstName == **null**) {

**throw** **new** IllegalArgumentException("Field cannot be empty.");

}

**else** **if** (firstName.length()>10) {

**throw** **new** IllegalArgumentException("Name must be less than 10 characters.");

}

**else** {

**this**.firstName = firstName;

}

}

**protected** **void** setLastName(String lastName) {

**if**(lastName == **null**) {

**throw** **new** IllegalArgumentException("Field cannot be empty.");

}

**else** **if**(lastName.length()>10) {

**throw** **new** IllegalArgumentException("Name must be less than 10 characters.");

}

**else** {

**this**.lastName = lastName;

}

}

**protected** **void** setNumber(String number) {

String regexTest = "[0-9]+";

**if**(number == **null**){

**throw** **new** IllegalArgumentException("Number cannot be null.");

}

**else** **if**(!(number.length()==10)) {

**throw** **new** IllegalArgumentException("Number must have a length of 10.");

}

**else** **if**(!(number.matches(regexTest))) {

**throw** **new** IllegalArgumentException("Only numbers are allowed for phone numbers.");

}

**else**{

**this**.number = number;

}

}

**public** **void** setAddress(String address) {

**if**(address == **null**){

**throw** **new** IllegalArgumentException("Address field may not be empty.");

}

**else** **if**(address.length()>30) {

**throw** **new** IllegalArgumentException("Address max length is 30.");

}

**else** {

**this**.address = address;

}

}

**public** **void** setContactID(String contactID) {

**if**(contactID == **null**) {

**throw** **new** IllegalArgumentException("ContactID cannot be null.");

}

**else** **if** (contactID.length()>10) {

**throw** **new** IllegalArgumentException("ContactID must be shorter than 10 digits.");

}

**else**{

**this**.contactID = contactID;

}

This piece of code was a little difficult to use but I found that it was more of a repeating process by the time it was done.

**Reflection:**

I did use different software testing techniques that I used were Junit and the regular testing techniques. I used manual testing. where I tested the code right before fully writing it just to ensure that I was on the right path. A technique that I did not use was nonfunctional testing because it was already integrated with the unit testing that was performed. Different projects require different types of testing. Our project needs more of a security type of testing, that is where the Unit testing was important. Manual testing is something that I am already familiar with so that was implemented as a routine step of mine. Nonfunctional testing has to do with security as well, but I didn’t think it was necessary to test both nonfunctional and unit at the same time, on the same project.

I applied caution when the testing was not coming together like I thought it would. I kept getting errors in the code and when I fixed the errors the test would fail. I came across a lot of failed tests. Writing the test code is really what had me on edge. For example, in my appointment test code I had to write the tests for the ID, date and description. This is where I had to use the interrelationship in my code for the unit test to even work correctly. I tried to be as simple as possible yet effective in writing the code. I wouldn’t have really been biased in the fact that “my code is better”. I would love some criticized feedback. I feel I would go a little more.

I do feel that It is very important to not cut corners in your code. Your code builds off of each other, so if you write bad code and errors are present the entire project will not test right or run properly. Cutting corners is not an effective way to build these projects and then testing techniques don’t cut corners with you. Applying your own skill and knowledge in this type of field really pays off. Using best practices in development is always best to use.