7-2 Project Two

CS-320-T4514

Martin Reyes

Southern New Hampshire University

April 24, 202

My unit testing approach for each of the three features was entirely aligned with the software requirements. The point of testing is to ensure that the code is working as intended. When writing the code for every service it was my intention to meet all requirements that were requested by the client. For example, the client requested that the appointment service have an appointment description that cannot be longer than fifty characters and not be left null. Therefore, I designed my tests to test exactly that. This is an example of my test that tested to make sure a description over fifty characters will not be accepted:

public void testDescription2() {

try {

String invalidDescription = "This appointment was scheduled for +”reasons but +but the appointment's description is much too long. +It is over the 50 character limit.";

appointment.setDescription(invalidDescription);

} catch (IllegalArgumentException message) {

assertEquals("Invalid. Must be less than 50 characters and not null", message.getMessage());

}

}

I can tell my tests were effective because of two reasons. One was that the tests successfully passed. The other is that at least the minimum required code coverage was achieved. Overall, I had a good experience writing junit tests. It was a new skill for me, and I have much to learn still but, I know it will come in handy as I advance my career in software development. I was able to ensure my code was technically sound and efficient by following the coding best practices. For example, my code was properly indexed and followed consistent naming conventions and comments to improve readability. I also kept the comments as concise as possible. This is the code for setting the date for an appointment:

//Check if date is not in past or null

public void setDate (Date appDate) {

//Create date and store

Date currentDate = new Date();

//Check if date is not in past or null

if ((!currentDate.before(appDate))) {

throw new IllegalArgumentException("Invalid.");

}

this.appDate = appDate;

}

Junit testing is very useful way of testing. As I have learned throughout the course, it is great for both the black box approach and the white box approach. I mostly used the white box approach in my testing as I knew the inner workings of my code and wanted to check the outputs produced. I did not use any experience-based testing techniques because I am still very new to testing but other than that, the requirements given were very clear and enough so that experience-based testing was not necessary. I kept my mindset as analytical as possible. This goes hand in hand with being cautious because it is important appreciate the complexity and interrelationships of the code being testing as it is my job as a tester to find errors in the code. The quality of the overall product will largely reflect on how good a job I do as a tester. If I let errors slip by, it can cost the company a lot of money and resources to fix. That is why it is important to pay attention to detail and not underestimate the code being tested. As a software engineering professional, it is important to stay disciplined in the commitment to quality. Cutting corners will almost certainly lead to having errors in the code and if not then it is very possible that parts of the requirements were not fully met. This can hurt the company in many ways. Errors will lose the company a lot of money and resources. If requirements are not fully met, then the client will be unsatisfied. The team will be made to complete what should have been already done or even rework the insufficient code. Sometimes an unreasonable deadline is unavoidable and is what causes technical debt; however, I will still try my best to never prioritize the quick completion of code over writing the best quality code.