Zheen H. Suseyi

06/17/2023

CS320

SNHU

Project One Summary and Reflection

**Summary**

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

There were three different programs that made up this software each with their own requirements. Contact Service required Junit testing of ID, firstName, lastName, phone number, address, and the ability to add, update, and delete contacts by ID. Task Service required Junit testing of ID, name, description, and the ability to add, update, and delete tasks by ID. The last program was Appointment Service. Appointment Service required Junit testing of ID, date, description, ability to add, and delete appointments by ID. All three of these programs not only required Junit testing but also the creation of these programs from nothing. My approach has always been extensive testing as I am writing the code and making sure my program is perfect before submission. One specific example of this is completely messing up my submission for Contact Service initially by not including my Junit tests in the program. I realized this with only 1 day left to submit Task Service and made sure to completely rework my Contact Service program and include my Junit Tests so that I can resubmit a revised version of Contact Service along with my Task Service.

***Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?***

I felt that my JUnit tests tested exactly what I wanted to be tested so I would say that they were very effective. When it came to the coverage percentage, I found that my coverage percentage would be low because of the setters I had in my main function. The issue was that those setters are required for functions in my ContactService and TaskService classes. However I felt that my coverage tested exactly what I needed to because I would extensively test and rerun each individual Junit test until I got exactly the desired result. Test coverage is the one area I couldn’t figure out because I really tried to get above 60% test coverage on my programs. Everytime I would try I couldn’t without deleting code that is essential for other functions so I decided to just focus on delivering a working program with function Junit tests instead.

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

One section of my code that is technically sound was making sure my date wasn’t in the past, here is how I did it:

*public Appointment(String ID, Date appointmentDate, String Description){  
 Date date = new Date(System.currentTimeMillis());*

*if(appointmentDate.before(date)) {*

*throw new IllegalArgumentException("Date is in the past and invalid"); }*

*}*

Now it was time to test to see if this worked using a Junit Test.

*Date date = new Date(System.currentTimeMillis());*

*Date appointmentDate = new Date(System.currentTimeMillis() + 500000);*

*Date pastDate = new Date(System.currentTimeMillis() - 500000);*

*// Testing if date is in the past*

*@Test*

*void testDateInPast() {*

*assertTrue(pastDate.before(date));*

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

*new Appointment("12345", pastDate, "John works in IT");*

*});*

*}*

It took me a while to figure out how to use Date in java and figuring out the logic behind getting this to work. In the end, I have this really great sequence of code that works according to my Junit test and is very technically sound.

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

One way I knew my code was efficient was how I checked for specific errors using Junit. In my Appointment Service program, I put nearly everything into the following constructor so that it was really easy to check for errors using Junit:

public Appointment(String ID, Date appointmentDate, String Description){

*// Check if id is null*

*if (ID == null) {*

*throw new NullPointerException("ID is required and cannot be null.");*

*}*

*// Check if id length exceeds 10 characters*

*if (ID.length() > 10) {*

*throw new IllegalArgumentException("ID cannot have more than 10 characters.");*

*}*

*// Check if appointment is null*

*if (appointmentDate == null) {*

*throw new NullPointerException("Date is required and cannot be null.");*

*}*

*// Check if appointment date is in the present*

*Date date = new Date(System.currentTimeMillis());*

*if(appointmentDate.before(date)) {*

*throw new IllegalArgumentException("Date is in the past and invalid");*

*}*

*// Checking if description is null*

*if (Description == null) {*

*throw new NullPointerException("Description is required and cannot be null.");*

*}*

*// Checking if description length greater then 50*

*if (Description.length() > 50) {*

*throw new IllegalArgumentException("Description cannot have more than 50 characters.");*

*}*

*this.ID = ID;*

*this.appointmentDate = appointmentDate;*

*this.Description = Description;*

*}*

Checking for errors in Junit was extremely easy because of this constructor. All I had to do was call it during my Junit tests, making my code extremely efficient.

**Reflection**

***What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.***

The software testing techniques I applied were essential to the completion of the project. The techniques used were Unit testing, Acceptance testing, and System testing. Unit testing was the main theme of the program because this project tested individual characteristics of each function which is exactly what unit testing is all about. Acceptance testing is determining whether our software is ready to be shipped out by testing the entire application which I did before every submission. System testing is similar to Acceptance testing in which the entire software is tested to see if all the requirements are met.

***What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.***

This project mainly used Junit to test individual requirements within the project, so I would say that Integration testing was not used in this project. Integration testing would be testing the entire software to see if everything works together as a unit which was never done. Other testing methods not used in this project would be Security testing which tests for security flaws, and Software performance testing which tests for efficiency in how the project runs.

***For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.***

Unit testing tests individual specific requirements within a program and is what I primarily did with Junit during this project. Acceptance testing is the testing phase right before a project gets shipped out by testing the entire project for submission. System testing is when the entire project is tested just to see if the requirements are met. The testing techniques not utilized but still very important would be firstly integration testing which determines if the software works together as a unit. Security testing was also not used at all despite being extremely important for the security of the company and users. Finally, Software performance testing was also not utilized mainly because this was a smaller program but if the program was really large i think this should definitely be utilized.

***Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.***

Throughout the entire project I made sure to exercise caution because one wrong line of code could set me back severely and make everything much harder. A specific example would be setting the date in my Appointment Service class:  
*// Check if appointment date is in the present*

*Date date = new Date(System.currentTimeMillis());*

*if(appointmentDate.before(date)) {*

*throw new IllegalArgumentException("Date is in the past and invalid");*

*}*

I had to make sure to check if the appointment date was in the past by making a new variable “date” be in the present. This part could have easily been messed up and could have made my entire project wrong. It's important to appreciate the complexity and interrelationships of the code that is being tested because one small syntax error or incorrect logic in the code can completely make a project stop working.

***Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.***

Naturally, any code I write is biased because I really work hard on anything I code and when something appears to be running and working I have a hard time changing it. One way I try to limit bias in review of my code is by looking really closely into the rubric and what is required of me and try and get it down to the very detail. I imagine that for a software developer, bias could be a bit of a problem if they have trouble following directions or like to do things in their own way. One example of how I was bias in my code was me implementing a getTaskbyID function instead of including it somewhere else:

*// Getting any task by its ID*

*public static Task getTaskById(String taskId) {*

*for (Task task : taskList) {*

*if (task.getTaskID().equals(taskId)) {*

*return task;*

*}*

*}*

*return null; // Task with the given ID not found*

*}*

***Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.***

Discipline is a skill that is honed throughout your entire life and is something that I think is truly a test of one's character. It is extremely hard being disciplined when there are so many distractions and negative influences around the average person these days and discipline can manifest in ones work. Staying disciplined in your code by submitting error-free code that been extensively tested while fulfilling the requirements that is needed is something that I have always strived for. I found out the hard way about cutting corners when it comes to coding because eventually you fall behind and have to commit way more hours in learning how to do it the proper way and this just adds way more unnecessary stress for no reason. The best way to avoid technical debt is to write clean code that is tested frequently and works every time. A specific example would be my test to see if my Junit is working that I implemented for every program:  
// Testing if junit is working

@Test

void testAppointment() {

Appointment appointment = new Appointment("12345", appointmentDate, "New appointment for monday");

assertEquals("12345", appointment.getID());

assertEquals(appointmentDate, appointment.getAppointmentDate());

assertEquals("New appointment for monday", appointment.getDescription());

}

This code may not be necessary but it gives me assurance that Junit is working so that I can focus on my actual Junit tests.