Marcus Smith

CS 320

8/9/2023

Summary and Reflections Report

When it comes to the testing approach for the following three services, contact service, task service, and appointment service each one asked for different software requirements that in turn worked together to provide a complete system. For all three services, it was asked for the ability to create a unique ID that then can be used to identify certain tasks, contacts, or appointments when called. For this, I was able to complete this by creating a function that would generate a random number and then add it to the ID list for each needed java class when the function was called. By doing this, when a new contact, appointment, or task needed to be created, this function would be used to allow for each one to be called upon if needed, and also allow for the deletion of said contact, appointment, or task if something like the completion of the task, the passing of the appointment, or the deletion of a contact that is no longer apart of the system needed to be made. For all three services, when it came to adding and deleting the needed information for each service, I created specific functions for each option that the customer wanted, and used illegal argument exceptions that would throw statements when the provide information either exceeded the character length amount or if the inputted data was null. If the input passes the needed requirements, then the function would then run and add or delete the needed parameters. Then for the Contact and Task Services, the ability to update the wanted parameters for the software, such as for the Contact Service allowing for the updating of the first and last name, as well as the phone number and address. For the Task Service, allowing for the task name and description to be updated for the software. When applying these multiple options, I used a case switch function to allow for the user to input the selected option they want to update and then input and apply the update.

For the Contact Service, Appointment Service, and Task Service to test the applied code, the functionality of the programs, I created JUnit tests. With these tests I used assertions to apply the function variables. By using assertTrue, I was able to create test inputs, and by then creating test to apply the wanted parameters, all having a test ID being created, and such as a first name and last name for the contact test, as well as the phone number and address. For the Task tests, creating the task name and task description. For the Appointment tests, by creating the date and description for the appointment. By then creating tests to test for length issues for the ID and descriptions, and testing for the date and whether it is before a valid date. When running these tests, I made sure that the test coverage was a hundred percent for all the tests. By ensuring that all the areas that needed to be incorporated in the software were tested, when all the tests pass with a hundred percent coverage, it allowed me to make sure that the JUnit tests were effective. To ensure that the code was technically sound, by using the assertion assertThrows and by applying parameters that would cause a throw such as,

@Test

**void** testAppointmentIdTooLong() {

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

**new** Appointment("12345543211", appointmentDate(), "Take appointment and print file.");

});

}

, I am able to test for an incorrect ID length and also make sure that the right response is gotten from the test. This strategy is also applied to other parameters, such as description length,

@Test

**void** testTaskDescriptionTooLong() {

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

**new** Task("1234554321", "Print", "Take task and print file to display and show the user that the file has been printed and should be ready to be used");

});

}

, as well as checking for the response with invalid dates for the appointments,

@Test

**void** testAppointmentDateBefore() {

Calendar c = Calendar.*getInstance*();

c.set(Calendar.***MONTH***, 6);

c.set(Calendar.***DATE***, 27);

c.set(Calendar.***YEAR***, 2023);

Date invalidDate = c.getTime();

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

**new** Appointment("1234554321",invalidDate, "Take appointment and print file.");

});

}.

To make the code more efficient, I created functions that could be applied to the different test is wanted to run such as in the appointment service tests, where I created a date that can then be applied to the adding of an appointment and deletion of an appointment,

//Creates a valid date to test with

Date appointmentDate() {

Calendar c = Calendar.*getInstance*();

c.set(Calendar.***MONTH***, 7);

c.set(Calendar.***DATE***, 27);

c.set(Calendar.***YEAR***, 2023);

Date date = c.getTime();

**return** date;

}

, just like the code above.

As I worked on the software of the project, I applied dynamic testing techniques, in which I ran tests and ran the code to determine the coverage of the tests. Doing multiple condition coverage tests, I aimed to identify the correct and incorrect outcomes of the conditions I created for the programs. By also following proper coding guidelines, I put comments and worked to make sure the code was clean and readable, for other peoples use. By also testing as I created the test code, I was able to make sure that each part worked when they are coded. Some software testing techniques that I did not use include some static testing techniques, such as reviewing between other developers/testers. The practical use of these techniques can help catch problems in the early stages of the software’s development. By also following proper coding etiquette helps to make the software accessible by many and allows those that review the software an easier time working through the code.

As I worked on this project, the mindset I adopted was to look to not just test correct outcomes but to also look to check for proper responses to incorrect outcomes. As I worked as a software tester, I made sure to employ caution in the fact that I worked to create the tests for the software, such as in not just testing if contact information met the criteria, but by also testing the getters as well as the setters. The importance of appreciating the complexity and interrelationships of the code as I tested it is highly important, as by understanding these things allows for proper testing and helps to guide the areas my code with hit on. Whether testing of length is important, testing of assigning of the contacts information to a uniquely generated ID number. When it came to me trying to limit my own bias in reviewing of my code, I looked to make sure that I was open to comments given after my sections of software were made, and then apply them to my code, changing my view on my work. Initially because of my personal look on the code I had issues creating the test for the contactServiceTest class I created for the Contact Service. After stepping back and researching and working on the next set of classes for the Task Service, I was then able to go back and apply code to allow for testing in the contactServiceTest.

All of this is important to helping stay disciplined in the quality of my software. Making sure to hit on as many outcomes in the software, while testing early and often to apply changes when need be. This can help create sound code, and to allow for easier access by other software testers/developers. By also being disciplined and writing code that follows the guidelines and making sure not to take shortcuts, though speed may be lost compared to taking the shortcuts in the code, the code will be sounder, allowing for my confidence in the release of the software when the time comes.

Resources:

Bose, S. (2023). Software Testing Techniques: Explained with Examples | BrowserStack. *BrowserStack*. https://www.browserstack.com/guide/software-testing-techniques

GeeksforGeeks. (2023). Software testing techniques. *GeeksforGeeks*. https://www.geeksforgeeks.org/software-testing-techniques/