Project 2

Reflection

James Carver

2022

The testing approach aligned with the software requirements because it tested all the

methods used to create the code. The contact class, task class, and appointment class each had requirements that the input could not be longer than a certain number of characters or had to be an exact number of characters. For example, firstName, lastName, contactID, taskID, and appointmentID all could not be longer than 10 characters, while phone had to be exactly 10 digits, the address could not be longer than 10, and both task and appointment descriptions could not be more than 50. The quality of the Junit tests improved with each integration of code I added. In the end, I achieved an 84.6% coverage of the whole project. With more time and effort, I believe it could be higher. There are some areas of testing I still do not understand or could not find the correct test for. The services and service tests all achieved 100% coverage results, while the others had various results. I attempted to duplicate code between each of the classes, but it still had different outcomes.

To ensure the code was technically sound, I implemented different coding techniques.

First, I broke the code down into sub-parts and then came up with a solution for each. For each piece of code I wrote, I created a test to accompany it each time running it. Once I found a piece of code that worked, I stuck with that style for the rest. For example, delete contact, delete task, and delete appointment all use the same algorithm.

public boolean deleteTask(String taskID) {

for (int i = 0; i < tasks.size(); i++) {

if (tasks.get(i).getID().equals(taskID)) {

tasks.remove(i);

System.***out***.println("Task deleted.");

return true;

}

if (i == tasks.size() - 1) {

System.***out***.println("Task ID: " + taskID + " not found.");

}

}

return false;

}

public boolean deleteContact(String contactID) {

for (int i = 0; i < contacts.size(); i++) {

if (contacts.get(i).getContactID().equals(contactID)) {

contacts.remove(i);

System.***out***.println("Contact deleted.");

return true;

}

if (i == contacts.size() - 1) {

System.***out***.println("Contact ID: " + contactID + " not found.");

}

}

return false;

}

public boolean deleteAppt(String apptID) {

for (int i = 0; i < appts.size(); i++) {

if (appts.get(i).getApptID().equals(apptID)) {

appts.remove(i);

System.***out***.println("Appointment" + apptID +" Deleted.");

return true;

}

if (i == appts.size() - 1) {

System.***out***.println("Appointment ID: " + apptID + " not found.");

}

}

return false;

}

To ensure that the code was efficient, the first thing I did was to make sure that each element was in its own function. Second, I made sure that there were no unnecessary operations. Last was the Junit tests. The Junit test ensured that the code functioned as required. For example, This test came back 100% for the getters and also showed that it was added to the array.

*@Test*

void testtContactClass() {

Contact contact1 = new Contact("1001", "John", "Wick", "5555555555", "100 Kick Ass Ln");

*assertTrue*(contact1.getContactID().equals("1001"));

*assertTrue*(contact1.getFirstName().equals("John"));

*assertTrue*(contact1.getLastName().equals("Wick"));

*assertTrue*(contact1.getPhone().equals("5555555555"));

*assertTrue*(contact1.getAddress().equals("100 Kick Ass Ln"));

}

The software testing techniques used for this project would be considered white-box

testing. White box testing is taken from specifications of what the system should do. In this case, the requirements of the assignment. This gives us an opportunity to inspect the inner working of the software. It tests for both valid and invalid inputs. The tests verified the code with path, statement, and branch coverage. Static analysis was also used for testing purposes which includes proof of correctness, code review, algorithm analysis, and code inspection.

I did not use black box testing. Black box testing is software testing where the internal structure of the program or the code is hidden, and nothing is known about it. I wrote the code so black box would not be used. Others that were not used include scripted box, dynamic analysis exploratory, manual, and automated testing.

White box is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software. Black box is a way of software testing in which the internal structure or the program or the code is hidden, and nothing is known about it. Dynamic is the execution of software for possible errors. This is better closer to the software completions or to test the functionality of a snippet of code.

During this project, I took caution in every aspect. This was the first time using Junit testing and only the second or third time using Java. Complexity and interrelationships were at the core of the code. Everything had to work together. If one function did not work, it would affect the other functions, at which point debugging becomes difficult. Because each of the objects had a particular function to perform, it relied on others to support it. I felt it was not hard to limit the bias of my code. I know I am new to coding and used many examples from different sources. While reviewing my code, I did many rewrites of different functions because I was skeptical that it would work altogether. For example, I was having trouble writing the Junit tests and found that writing the functions as a Boolean and getting a return helped with the testing. I could see later in my career being biased toward my own code. This week’s discussion on errors in code and the damage they can cause shows the importance of being disciplined in quality. Cutting corners could cost money, ruin reputations, or even take lives. I believe asking questions, taking my time, doing it right the first time, and always growing will help me avoid technical dept.

References

Hambling, Brian, et al. *Software Testing : An ISTQB-BCS Certified Tester Foundation guide –*

*4th edition*, edited by Brian Hambling, BCS Learning & Development Limited, 2019.*ProQuest Ebook Central*, <http://ebookcentral.proquest.com/lib/snhu-ebooks/detail.action?docID=5837074>.