**Summary and Reflections Report**

My unit testing approach for each of the three features of this project was first to ensure that every aspect of the software requirements document was satisfied. I accomplished this by analyzing the requirements line by line and implementing each one at a time. Once I met the requested software goals within the product, I formulated adequate JUnit tests to cover various scenarios. Specifically, after adding an object, I checked that its attributes were not null and ensured the program would throw an exception if the attributes were too long. This practice is evident in sections of code such as,

“public Contact(String id, String firstName, String lastName, String number, String address) {

if(id == null || id.length()>10) {

throw new IllegalArgumentException("Invalid ID");

}

if(firstName == null || firstName.length()>10) {

throw new IllegalArgumentException("Invalid First Name");

}

if(lastName == null || lastName.length()>10) {

throw new IllegalArgumentException("Invalid Last Name");

}

if(number == null || number.length()>10) {

throw new IllegalArgumentException("Invalid Number");

}

if(address == null || address.length()>30) {

throw new IllegalArgumentException("Invalid Address");

}

this.id = id;

this.firstName = firstName;

this.lastName = lastName;

this.number = number;

this.address = address;

}”

My JUnit tests were effective based on the overall coverage percentage of 84.5%. Upon further inspection, the areas not covered by the JUnit tests were the loops that I implemented to check if an ID was in use, the creation of a vector to store objects, and adding the object to the vector. My “add” method test checks if an object at position 0 has the expected attributes and therefore checks if the object was created and added to the vector. This testing method also ensures the construction of the vector. To ensure that my code was technically sound, I utilized assertions such as,

“assertTrue(service.tasks.elementAt(0).getTaskId().equals("1234567890"));

assertTrue(service.tasks.elementAt(0).getTaskName().equals("Julie"));

assertTrue(service.tasks.elementAt(0).getDescription().equals("Adams"));”

I used the least amount of code possible to ensure my code was efficient while including all requirements. For example,

if(id == null || id.length()>10) {

throw new IllegalArgumentException("Invalid ID");

}

if(firstName == null || firstName.length()>10) {

throw new IllegalArgumentException("Invalid First Name");

}

if(lastName == null || lastName.length()>10) {

throw new IllegalArgumentException("Invalid Last Name");

}

if(number == null || number.length()>10) {

throw new IllegalArgumentException("Invalid Number");

}

if(address == null || address.length()>30) {

throw new IllegalArgumentException("Invalid Address");

}

I used a vector for memory storage since they can fluctuate in size, a desirable quality when the maximum size of the data structure has yet to be determined. Regarding efficiency in testing, I achieved 100% coverage in the Task and Contact classes.

**Testing Techniques**

The types of testing utilized in this project include functional JUnit unit testing, regression testing, and static testing in the form of visual review. An example of this would be,

@Test

void apptNotInPast() {

Calendar myCalendar = new GregorianCalendar(2021, 12, 10);

Date myDate = myCalendar.getTime();

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

new Appointment(“1234567890”, myDate, “Murray”);

});

}

Regression testing was performed after changes were made to ensure that new faults were not created.

Forms of testing not implemented in this project are maintenance, integration, system, and performance testing. Maintenance testing to performed on software that has already been released after updates or other changes are made. Integration testing tests the interaction of a few smaller sections of code after one area has been unit-tested. System testing takes integration testing to a system-wide scale, checking the entire program for proper functionality. Performance testing tests a full product near completion to ensure it meets customer needs. Non-functional testing, which includes usability, performance, efficiency, and security, was also not utilized in this project (Hambling et al., 2019, p. 66).

**Mindset**

In acting as a software tester, I employed significant caution. Appreciating the complexity and interrelationships of the code being tested to form adequate tests is vital. Failure to analyze these relationships within a piece of software will result in insufficient testing, false negatives, more time invested in development and testing, and a significant increase in costs associated with the project. Bias is also an issue, especially when testing your code, since one is always inclined to believe they have written the perfect piece of software. Very rarely if ever is this the case. Far more often than not, bugs and faults were overlooked during development. The mindset I approach with is that there ARE flaws present, and it is my job to break this code. I consider how often I must go back and make changes to sections while developing and recognize that I am not beyond reproach. We must remain vigilant and exercise discipline in testing code, especially if it is our own, to overcome bias and complacency. People will inevitably cut corners, so approaching with a mentality of being conscious of this fact and your own bias will save a great deal of time and money for you, your employer, and your client.

**References**

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). Software testing : An istqb-bcs certified tester foundation guide - 4th edition. BCS Learning & Development Limited.