Brent Longstreet

Professor Kraya

CS-320

October 12, 2023

Mobile Application Summary & Reflection Report

**Summary**

I analyzed and performed tests on the mobile application I created. Each of the three features utilized a testing approach known as white-box testing. This unit-testing approach allows the tester, me, to conduct tests with knowledge of the source code. After analyzing the three features, I consider my testing approach to be aligned strictly with the software requirements. I ensured the unique ID is not longer than 10 characters, is not null, and cannot be updated for both. Each project had its own unique variables and requirements. I created these variables and helper functions with the consideration of the length, data type, scope, and other requirements in mind. Test functions were created for both the contact and service project to confirm the deletion or insertion of a task or contact worked successfully.

**Function to update appointment id and ensure the length is <= 10 and id is not null**

public void updateAppointmentId(String id) {

//throw error if appt id is null

if (id == null) {

throw new IllegalArgumentException("Appointment ID cannot be null.");

} else if (id.length() > 10) { //throw error if appt id length is > 10

throw new IllegalArgumentException("Appointment ID cannot exceed " +

10 +

" characters.");

} else {

//update appt id

this.appointmentId = id;

}

}

**Function to set the description of a task and ensure the length of the description is <= 50 and is not null**

protected void setDescription(String taskDescription) {

//check if description is empty or greater than 50 characters

if (taskDescription == null || taskDescription.length() > 50) {

throw new IllegalArgumentException(

"Task description is empty and/or greater than 50 characters");

} else {

//valid description, set description

this.description = taskDescription;

}

}

The Junit tests were effective on the basis of coverage percentage due to the variance of which the tests checked. I ensured each feature has an 80% or greater coverage passage to consider it a quality test. The test aligned closely with the software requirements to ensure the functionality and accuracy of the software. The tests verified each field of the task and contact was equal to the value that was passed in. It verified the fields that shall not be null were not null. The input length of specific fields was within the maximum bounds specified. Finally, it checks that a new value cannot be entered into fields that are not updateable.

This code example shows the Junit tests conducted when creating a new task. The name variable passed into the new tasks is a string that consists of 20 characters. I check to verify the task is not null and that the name that was passed in is equal to what is returned when retrieving the name. This ensured that my code was technically sound by accounting for the software requirements provided. I test the maximum length bounds of the name, ensure it is not null, and that the name retrieved is equal to the name passed in.

@Test

  void newTaskNameTest() {

    TaskService service = new TaskService();

    service.newTask(name);

    Assertions.assertNotNull(service.getTaskList().get(0).getName());

    Assertions.assertEquals(name, service.getTaskList().get(0).getName());

  }

The code example shows my code was efficient. I utilized getter and setter methods for each of the fields in the contact project. This ensures the first name passed in follows the software requirements. The first name’s length must be less than or equal to 10 and it cannot be null. Utilizing a getter and setter allows me to validate the passed-in value passes the requirements. Additionally, using getters and setters will make future implementations of error handling easier.

 //setter methods

  public void setFirstName(String fName) {

    if(fName.length() <= 10 && fName != null) {

      this.firstName = fName;

    }

  }

  //getter methods

  public String getFirstName() {

    return firstName;

  }

**Reflection**

**Testing Techniques**

Some software testing techniques I employed in this project are listed below.

**Regression Testing:**

Regression testing is performed after the code has been updated to try and locate new bugs caused by the update. It is most effective for software that is always changing, as regression testing runs after each change. This testing technique can test both functional and non-functional requirements.

**Integration Testing:**

Integration testing combines individual components of software together to test its functionality after being integrated. The integration process can be gradual to locate bugs that arise from each integration. Integration testing typically operates in a specific order. These tests are performed in sequential order and new individual components are added after successfully integrating the previous.

**System Testing:**

System testing involves analyzing and testing the program as a whole. It ensures the program meets the client's requirements. System testing prioritizes verifying the functionality of the application.

Some software testing techniques I didn’t use are performance testing and usability testing. Performance testing ensures the programs operate at functional speed, are scalable, and are stable. Usability testing ensures the program is user-friendly and intuitive for the user.

The practical use of unit testing is to ensure the functionality of the code. It involves automated tests that take place early in the software development life cycle. This testing technique verifies the code’s logic is valid and meets the requirements. Unit tests are sometimes written before the code. This is called test-driven development. It enforces the implementation of proper tests before coding the software.

**Regression Testing:**

The practical use of regression testing is to locate newly created bugs after updating the software’s code. It helps to identify bugs early after implementing new implementations to easier identify them and their source.

**Integration Testing:**

The practical use of integration testing is to validate the pairing of multiple components and their functionality as a collective unit. It helps to identify potential errors and bugs caused after integrating new components.

**System Testing:**

The practical use of system testing to ensure the program meets the client’s requirements by testing the software as a whole. It verifies functionality and the completed software is acceptable for the user.\

**Mindset**

I adopted the mindset of a software tester while working on this project. It is not a mindset I was used to and took some time to adjust. I am able to now write code and perform tests on it. I employed the same caution I would if I worked for a business as a tester. I wanted the unit tests to have a high coverage and the software to meet the software requirements. I used getter and setter methods and prioritized the scope of variables and methods. I tried to remain unbiased by analyzing the tests performed for faults and coverage percentage. Bias would be a huge concern on the software developer side if they were testing their own code because it would lead to impartial and erroneous. Testing. When writing test it is important to not cut corners and accurately test all aspects of the software to ensu The practical use of unit testing is to ensure the functionality of the code. It involves automated tests that take place early in the software development life cycle. This testing technique verifies the code’s logic is valid and meets the requirements. Unit tests are sometimes written before the code. This is called test-driven development. It enforces the implementation of proper tests before coding the software.

**Regression Testing:**

The practical use of regression testing is to locate newly created bugs after updating the software’s code. It helps to identify bugs early after implementing new implementations to easier identify them and their source.

**Integration Testing:**

The practical use of integration testing is to validate the pairing of multiple components and their functionality as a collective unit. It helps to identify potential errors and bugs caused after integrating new components.

**System Testing:**

The practical use of system testing to ensure the program meets the client’s requirements by testing the software as a whole. It verifies functionality and the completed software is acceptable for the user.

re its validity. Not doing so will lead to faulty and inaccurate testing.