Devin Criswell

6/21/2024

Summary and Reflection

CS 320

My approach was to align my software directly with the provided requirements. I read through the requirements thoroughly before starting development. I also referred back to the requirements during the development process to ensure that the software was inline with the requirements. I utilized testing as a way to make sure my software was meeting the expectations of the software.

JUnit testing was useful for providing validation that the software was meeting the requirements. Conducting tests looking for nulls or length helps to ensure the requirements are being met. With the contact ID the string could not be longer than 10 characters or null. Creating a test for this and having an error thrown allows the software to be compliant with the requirements. I tried to create my tests based around the requirements and looked for results that provided feedback indicating I was getting the correct coverage. If I got those results, then it confirmed I was using the proper JUnit tests.

To ensure that my code was technically sound I implemented checks into the constructors of the classes. These were helpful in ensuring that the required constraints were met as per the requirements. The JUnit tests verified this and the functionality for the service classes.

public Contact(String contactId, String firstName, String lastName, String phone, String address) {

if (contactId == null || contactId.length() > 10)

throw new IllegalArgumentException("Invalid contactId");

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

throw new IllegalArgumentException("Invalid firstName");

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

throw new IllegalArgumentException("Invalid lastName");

if (phone == null || phone.length() != 10)

throw new IllegalArgumentException("Invalid phone");

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

throw new IllegalArgumentException("Invalid address");

this.contactId = contactId;

this.firstName = firstName;

this.lastName = lastName;

this.phone = phone;

this.address = address;

For ensuring efficiency I used HashMap for storing and managing the objects. This works well for these requirements and provides constancy throughout the application. The tests also ensure that the software meets the requirements by testing for different scenarios and edge cases. This helps to validate how technically sound and efficient the code is.

public class ContactService {

private final Map<String, Contact> contacts = new HashMap<>();

public void addContact(Contact contact) {

if (contacts.containsKey(contact.getContactId())) {

throw new IllegalArgumentException("Contact ID already exists");

}

contacts.put(contact.getContactId(), contact);

}

public void deleteContact(String contactId) {

contacts.remove(contactId);

}

public Contact getContact(String contactId) {

return contacts.get(contactId);

}

}

@Test

public void testAddContact() {

Contact contact = new Contact("1234567890", "John", "Doe", "1234567890", "123 Main St");

service.addContact(contact);

assertNotNull(service.getContact("1234567890"));

}

@Test

public void testAddContactDuplicateId() {

Contact contact1 = new Contact("1234567890", "John", "Doe", "1234567890", "123 Main St");

Contact contact2 = new Contact("1234567890", "Jane", "Smith", "0987654321", "456 Elm St");

service.addContact(contact1);

assertThrows(IllegalArgumentException.class, () -> {

service.addContact(contact2);

});

}

For this project I implemented JUnit for testing. JUnit was used for testing the classes. The constructors were tested to verify they would throw exceptions for invalid inputs. CRUD operations were tested to ensure that data was being handled correctly. Edge cases were tested to ensure the application could handle the limits of acceptable inputs. This was done by checking contact ID, firstName, lastName, phone, and address fields adhere to their length constraints. Testing for duplicates was performed to verify exceptions are thrown.

When I started the project, I had very limited experience testing software. I was used to letting the IDE run the software and fix any errors that occurred, this was my understanding of testing. It has become very obvious that just because the software runs does not mean that it is correct or that it is taking edge cases into account. While basic debugging is still important to the development process there is more that needs to be tested for. I was cautious during testing by continuously referring to the requirements to ensure that I was not missing anything. I also tested the software for edge cases. To limit my biases, I actively tried to separate the roles of developer and tester. As a tester, focusing on breaking the code and identifying flaws without considering the effort involved in fixing them. I tested my code often to ensure that my code was of quality and tried to follow industry standards to the best of my ability. Clean quality code is something that I have noticed instructors continue to preach to students. I also reviewed the requirements multiple times to ensure the software was meeting them, which adds to the quality of the final product.