Module 7 Project 2

Brandon Barrett

Southern New Hampshire University

Professor Omar Toledo

2/24/2024

While creating the application for project one, I used a systematic approach to unit testing. I ensured each feature was tested rigorously to align with the software requirements and to maintain high-quality code. I created unit tests that covered various scenarios. One example is in the Task class where I ensured that the full name attribute couldn’t be null, exceed 20 characters, or be an empty string. This validated both valid and invalid inputs against the requirements.

I used Unit testing to verify that each unit of code functions correctly as expected. For each of the features, I used Junit testing. I wrote test cases to validate the functions of specific classes and methods to ensure that they met requirements. In the first feature, I created code to allow changes to be made to contacts. In the second feature, I made code to test the ability to add, delete, and update tasks. In the third feature, I made code to add and delete appointments. Unit testing was important for all three assignments because it identified defects early and made debugging and maintenance easier.

I was able to ensure that critical parts of the code were thoroughly tested by targeting 100 percent coverage of getters, setters, and constructors. However, I didn’t test every possible constructor permutation, I was able to ensure that both valid and invalid constructor inputs were handled. To ensure that technical soundness was present in the application rigorous testing was conducted to validate various functionalities. For example, in the ‘testAddContact()’ method, assertions were made to confirm the correct addition of a contact which verified attributes such as first name, last name, phone number, and address:

assertEquals(firstName, ContactService.contactList.get(0).getFirstName()); assertEquals(lastName, ContactService.contactList.get(0).getLastName()); assertEquals(phoneNumber, ContactService.contactList.get(0).getPhoneNumber()); assertEquals(address, ContactService.contactList.get(0).getAddress());

This demonstrates the thorough testing approach applied to ensure that the contact information is accurately stored and retrievable.

I was able to achieve efficiency by aiming to optimize performance by using direct access methods and avoiding unnecessary iterations. One example of that is the ‘testEditBadAddress()’ method which validates the absence of an address update for an invalid ID:

assertFalse(testBool);

This confirms that the address remained unchanged when attempting to edit a contact with an invalid ID.

While developing this application there were a variety of software testing techniques that I implemented. One of which of course was the use of unit testing. Throughout this project I utilized Junit testing, which was used to verify the correctness under different conditions. For example, in the ContactService class, numerous methods were written to validate functionalities such as adding a contact, deleting a contact, and editing contact details. Static testing techniques were also used in this project to analyze the codebase for defects and adherence to coding standards. One technique that was not used but could be useful is exploratory testing. This is used to uncover unforeseen defects and usability issues. Exploratory testing involves the exploration of the functionalities of the application without predefined test cases. This is most useful in more complex applications and can also help identify potential issues that may have not been considered during development.

While going through the milestone assignments I made a lot of mistakes and I honestly had to start the project from scratch and take a completely different route, and I am glad I did. While staying more cautious within my code I was able to get above a 90% in my tests and it truly worked out in the long run. Caution is important throughout the testing process and I was able to ensure comprehensive validation of the application’s functionalities. Potential defects and vulnerabilities were identified early and I was able to minimize the risk of potential issues.

It is important to appreciate the complexity of the code being tested because it helps to understand the interrelationships and potential impact on the application as a whole. An understanding of data flow and manipulation was required in testing the interaction between the ContactService class and the data structures. The complexity of the code being tested also allowed for the consideration of non-functional requirements, such as performance and scalability.

I was able to utilize a critical mindset and was able to identify potential blind spots and oversights. By limiting bias in testing the codebase, I ensured objectivity and impartiality in reviewing the quality and correctness. The utilization of the Junit tests held me accountable in my own right. Having a bias in my testing is what I struggled with in the beginning of this class, however, I think I have grown from then and was able to succeed in this project. Being disciplined in the commitment to quality as a software engineer should be a priority of all software developers. People have lost their lives due to human error, which is why rigorous testing is pushed on developers so harshly. When it comes to my personal work ethic and how I would avoid technical debt, I have a moral code that I live by and if I believe that I am not producing top quality products I will be very vocal about it in the workplace. I think it is important for developers to be honest and very open about where they are during the development process to allow for additional assistance from their peers.

The mindset during testing plays a critical role in shaping the effectiveness of the testing process. This ultimately influences the success of software development projects.