CS 320 Project Two: Summary and Reflections Report

Name: Jose Medina  
Course: CS 320 Software Test, Automation QA  
Professor: Kalysa Wilson  
Date: 12/12/2024

Summary

1. Unit Testing Approach Aligned to Software Requirements

For this project, my unit testing approach focused on meeting specific requirements for the ContactService, TaskService, and AppointmentService milestones. Each service required clear constraints and validations to ensure data integrity:

* ContactService: I validated unique contact IDs, first and last name constraints (≤ 10 characters), phone numbers (10 digits), and address fields (≤ 30 characters).
* TaskService: Tests ensured unique task IDs, name fields (≤ 20 characters), and descriptions (≤ 50 characters).
* AppointmentService: I implemented unique appointment IDs, validated the date field to prevent past dates, and ensured the description was ≤ 50 characters.

JUnit tests were written to ensure these conditions were consistently verified. For instance:

* Adding a duplicate ID or invalid input triggered exceptions.
* Update tests validated modifiable fields like names or descriptions.
* Deletion tests ensured correct removal of data objects.

This approach directly aligned with the software requirements, as demonstrated by passing JUnit tests and adherence to validation constraints.

2. Effectiveness of Tests

The JUnit tests effectively validated all requirements for the services:

* Coverage Percentage: I ensured that the key functionalities—add, update, and delete operations—were fully tested.
* Edge Cases: Tests covered scenarios such as invalid input (e.g., null values, exceeding character limits) and duplicate IDs.

By verifying both positive and negative cases, the tests ensured the application would behave as expected under various conditions. The green results in Eclipse IDE confirmed the effectiveness of the tests.

3. Technically Sound and Efficient Code

* Technically Sound: I incorporated proper exception handling and input validation (e.g., ensuring task IDs are unique, validating date fields in AppointmentService).
* Efficient Code: I designed clean and modular code following object-oriented principles. Methods in each service (e.g., addTask, updateContact, deleteAppointment) were well-structured and concise.

For instance, in TaskService.java:

public boolean addTask(Task task) {

if (!tasks.containsKey(task.getTaskID())) {

tasks.put(task.getTaskID(), task);

return true;

}

return false;

}

This code is simple, readable, and efficiently checks for duplicate IDs.

Reflection

1. Testing Techniques Used

The primary testing technique used in this project was unit testing with JUnit 5. This involved:

* White-box Testing: Testing individual methods for logic errors (e.g., boundary conditions).
* Positive and Negative Testing: Ensuring both valid inputs and invalid inputs (null values, limits) behaved as expected.

2. Other Testing Techniques

While I focused on unit testing, I did not use techniques like:

* Integration Testing: Verifying interactions between multiple components or classes.
* System Testing: Testing the full application with external dependencies or interfaces.

These techniques were beyond the scope of this project but would be essential for a real-world application.

3. Practical Uses and Implications

The testing techniques ensured early identification of defects in logic and data validation. For example:

* If input constraints were not enforced (e.g., allowing null descriptions), the application might crash or produce unreliable results.
* Robust unit testing builds confidence in the application’s behavior and prevents issues from escalating during later development phases.

4. Mindset Adopted

* Caution: I approached the code cautiously to ensure all edge cases were accounted for. For instance, I included tests to prevent null or invalid data inputs. This ensured the reliability of each service.
* Bias Awareness: I was mindful of not assuming the correctness of my code. Running tests repeatedly and addressing failed cases helped improve my code’s quality.
* Discipline: Maintaining consistent and organized testing throughout the project helped me manage tasks efficiently and meet deadlines.

Example of disciplined testing:

* Each test case targeted a single requirement.
* JUnit results were reviewed after every iteration to ensure no regressions.

Conclusion

This project taught me the importance of writing efficient, technically sound code and validating it through thorough unit testing. While unit testing was the focus, I now understand the value of other testing techniques like integration and system testing for larger applications. Adopting a mindset of caution, avoiding bias, and practicing discipline allowed me to build reliable, maintainable, and well-tested code.