Mohamed ElhassanSouthern New Hampshire University  
mohamed.elhassan@snhu.edu  
CS320 - Software Test Automation QASummary and Reflections on Unit Testing for Appointment Management Services

### 

### **Summary and Reflections Report**

**Scenario:**

As a software engineer at Grand Strand Systems, I recently completed a project where I developed and tested back-end services for a mobile application, specifically focusing on the ContactService, TaskService, and AppointmentService. This report provides a summary of my unit testing approach, reflects on the effectiveness of the JUnit tests I wrote, and discusses the testing techniques and mindset I adopted during the project.

### 

### **Summary**

**Unit Testing Approach**

For each of the three features—ContactService, TaskService, and AppointmentService—I applied a systematic unit testing approach:

1. **ContactService**: The unit tests for ContactService were designed to verify the correct handling of contact creation, updating, and deletion. I wrote tests to ensure that invalid inputs were rejected, and valid operations behaved as expected.
2. **TaskService**: For TaskService, the unit tests focused on the management of tasks, ensuring that tasks could be added, updated, and marked as complete or incomplete. Tests also validated the handling of edge cases, such as attempting to update a non-existent task.
3. **AppointmentService**: The unit tests for AppointmentService checked that appointments were correctly scheduled, modified, and deleted. Special attention was given to overlapping appointments.

**Alignment with Software Requirements**

My approach was closely aligned with the software requirements. For instance, the requirements specified that the ContactService must reject contacts without a valid phone number, which I verified through multiple test cases. The TaskService required that tasks be marked complete or incomplete, a functionality I rigorously tested to ensure compliance. Similarly, the AppointmentService's requirement to prevent overlapping appointments was thoroughly validated.

**Quality of JUnit Tests**

The quality of my JUnit tests can be defended based on the coverage percentage achieved. With over 80% test coverage across the services, my tests effectively validated the majority of the code paths. This high coverage, combined with specific test cases targeting edge scenarios, ensures that the tests are both comprehensive and reliable.

**Experience Writing JUnit Tests**

Writing JUnit tests was an iterative process that involved identifying critical functionalities, developing test cases, and refining them based on the results. To ensure that my code was technically sound, I frequently reviewed the code structure, focusing on the proper use of assertions and avoiding unnecessary dependencies. For example, in the TaskService tests, I utilized parameterized tests to efficiently validate multiple input scenarios (e.g., @ParameterizedTest with @ValueSource for different task descriptions).

To ensure efficiency, I minimized redundancy in the test code. For instance, I used setup methods annotated with @BeforeEach to initialize common objects, which reduced code duplication and improved maintainability.

### 

### **Reflection**

**Testing Techniques**

During this project, I primarily employed **black-box testing** and **white-box testing**:

* **Black-box Testing**: This technique involved writing tests based on the software requirements without considering the internal code structure. It was particularly useful for testing the services' responses to various inputs, such as invalid contact information or overlapping appointments.
* **White-box Testing**: I used this technique to ensure that internal paths, conditions, and loops were tested. For example, in the TaskService, I tested all conditional branches to confirm that each scenario (e.g., task completion, task update) was handled correctly.

**Other Testing Techniques Not Used**

* **Integration Testing**: Although not directly applied in this project, integration testing involves testing combined components to verify their interactions. This technique would have been useful if I were testing how the ContactService, TaskService, and AppointmentService interacted with each other.
* **Regression Testing**: This technique, which involves retesting the application after changes to ensure existing functionalities are not broken, was not extensively applied. However, it would be crucial for future updates to the services.

**Practical Uses and Implications**

Each technique has its place in software development. Black-box testing is essential for ensuring compliance with requirements, while white-box testing ensures the internal code logic is sound. Integration and regression testing are vital for maintaining software quality in larger, more complex systems where components interact extensively.

**Mindset**

Throughout this project, I adopted a mindset of caution and thoroughness. I recognized the importance of appreciating the complexity and interrelationships within the code. For example, while testing the AppointmentService, I considered how changes in one part of the service might impact other functionalities (e.g., appointment overlap detection). This cautious approach helped prevent unintended side effects.

To limit bias, I frequently revisited my test cases from a fresh perspective, questioning assumptions I made during development. For instance, I revisited my tests for edge cases in the ContactService to ensure that I wasn't subconsciously avoiding potential flaws in my code.

As a software engineering professional, I recognize the importance of discipline in maintaining high code quality. Cutting corners, such as skipping tests for "obvious" cases, can lead to technical debt. To avoid this, I plan to adopt a rigorous approach to testing, ensuring comprehensive coverage and adhering to best practices, such as consistent code reviews and automated testing pipelines.

### 

### **Conclusion**

The experience of developing and testing the ContactService, TaskService, and AppointmentService has reinforced the importance of a disciplined approach to software testing. By carefully aligning my tests with the software requirements, employing effective testing techniques, and maintaining a vigilant mindset, I was able to ensure the delivery of high-quality code that meets the customer's needs.