**Summary and Reflections Report**

**Grand Strand Systems – Project One**

**Summary**

Unit Testing Approach for Each Feature

For this project, I developed unit tests for the ContactService, TaskService, and AppointmentService classes. Each feature had its unique requirements, and my testing approach aimed to validate core functionalities while checking for edge cases and invalid operations. According to ISTQB standards, effective unit tests should verify individual components in isolation before integration (Black & Van Veenendaal, 2022, Chapter 3).

* **ContactService:**  
  I tested adding, deleting, and updating contact records. The tests ensured that duplicate contact IDs triggered exceptions and that updating or deleting non-existent contacts behaved as expected.
* **TaskService:**  
  The unit tests validated adding tasks, preventing duplicates, deleting tasks, and updating both the task name and description. Tests for operations on non-existent tasks verified exception handling.
* **AppointmentService:**  
  Tests included adding appointments, preventing duplicates, deleting appointments, and verifying constraints on appointment dates and descriptions. The tests ensured that adding past-dated appointments or duplicate appointment IDs triggered appropriate exceptions.

**Alignment With Software Requirements**

My testing approach aligned closely with the software requirements, with each feature requirement translated into corresponding test cases covering both normal and edge conditions. According to the ISTQB Foundation syllabus, this is a critical aspect of confirming requirement coverage (Black & Van Veenendaal, 2022, Chapter 1).

For example:

* In TaskServiceTest, the testAddTask() confirmed that adding a duplicate task ID resulted in an IllegalArgumentException (JUnit 5 User Guide, n.d.).
* In AppointmentServiceTest, the helper method futureDate() creates future dates to enforce the requirement that appointments not occur in the past.

**Quality of JUnit Tests**

The high quality of the JUnit tests was evident in exceeding the 80% code coverage threshold. Tests validated both functionality and defensive coding practices. The ISTQB syllabus highlights the importance of verifying error handling and invalid inputs during component testing (Black & Van Veenendaal, 2022, Chapter 3). For example:

assertThrows(IllegalArgumentException.class, () -> service.updateTaskName("999", "New Name"));

This confirmed both correctness and resilience (JUnit 5 User Guide, n.d.).

**Experience Writing the JUnit Tests**

Writing these tests reinforced **test-driven development (TDD)** practices and the structured application of unit and exception testing as defined by ISTQB principles (Black & Van Veenendaal, 2022, Chapter 4). I structured tests with consistent @BeforeEach setup methods, used helper methods like futureDate() for reliable test data, and logically grouped related tests.

**Reflection**

**Testing Techniques Employed**

I primarily used **unit testing** and **exception testing**, both covered under component and negative testing approaches in the ISTQB syllabus (Black & Van Veenendaal, 2022, Chapter 3):

* **Unit Testing:**  
  Verified individual methods in isolation with both valid and invalid data (JUnit 5 User Guide, n.d.; Black & Van Veenendaal, 2022, Chapter 3).
* **Exception Testing:**  
  Confirmed proper handling of invalid operations, such as adding duplicate records or updating non-existent IDs.

**Other Techniques Not Used**

As noted in the ISTQB syllabus (Black & Van Veenendaal, 2022, Chapters 1–2):

* **Integration Testing:** Not used since services operated independently.
* **System Testing:** Not applicable for back-end service logic.
* **Performance Testing:** Not employed due to the project's limited scale.

Each of these techniques would be vital in larger systems, where integration and performance play critical roles.

**Mindset and Caution in Testing**

Following ISTQB recommendations, I maintained a careful, detail-oriented mindset, emphasizing error detection and defect prevention (Black & Van Veenendaal, 2022, Chapter 2). I tested scenarios like deleting an appointment twice to confirm no unintended side effects.

**Limiting Bias in Code Review**

I wrote tests after service logic was complete, then reviewed both critically, following ISTQB’s suggestion to diversify perspectives and proactively seek defects (Black & Van Veenendaal, 2022, Chapter 1). For instance, ensuring a null task name threw an exception.

**Commitment to Quality and Avoiding Technical Debt**

Consistent with ISTQB best practices (Black & Van Veenendaal, 2022, Chapter 4), I’m committed to test-driven development, exhaustive test coverage, and ongoing code improvement to avoid technical debt.

**References**

Black, R., & Van Veenendaal, E. (2022). *Software testing: An ISTQB-BCS certified tester foundation guide* (5th ed.). BCS Learning & Development Limited.

JUnit 5 User Guide. (n.d.). Assertions. Retrieved from <https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions>