**CS-320: Software Test Automation and QA**

**Thanh Nguyen**

**Instructor: Prof. Indigo Guerra**

**Project 2: Summary and Reflections Report**

1. **Summary**

* Unit Testing Approach for Each Feature:
* **Contact Service:**

Approach: Implemented JUnit tests focusing on unique contact IDs, valid phone numbers, and adherence to character limits.

Alignment to Requirements: Aligned closely with software requirements. For example, tests verified the uniqueness of contact IDs and proper format of phone numbers.

Quality Defense: Coverage exceeded 80%, ensuring comprehensive testing of all functionalities, including adding, updating, and deleting contacts.

* **Task Service:**

Approach: Employed JUnit tests concentrating on unique task IDs, valid names, and descriptions.

Alignment to Requirements: Aligned precisely with requirements, covering unique task IDs and proper validation of name and description fields.

Quality Defense: Maintained a coverage percentage above 80%, demonstrating thorough testing of task service functionalities.

* **Appointment Service:**

Approach: Developed JUnit tests checking for unique appointment IDs, valid dates, and proper descriptions.

Alignment to Requirements: Aligned perfectly with requirements, ensuring unique appointment IDs and validation of appointment dates.

Quality Defense: Achieved over 80% coverage, validating the effectiveness of tests in scenarios like adding and updating appointments.

* **Experience**

Writing JUnit tests for Project One was a crucial and insightful process. It involved meticulous planning, thorough understanding of software requirements, and constant iteration to ensure comprehensive coverage of functionalities. Each test aimed to simulate various scenarios, validating both expected and unexpected inputs.

Ensuring Technical Soundness:

To guarantee technical soundness, I followed best practices in writing JUnit tests. The focus was on covering edge cases, validating inputs, and ensuring proper exception handling.

A screen shot of a computer program

Description automatically generated

Ensuring Code Efficiency:

Efficiency in code is critical for optimal performance. While writing JUnit tests, considerations for efficiency involved avoiding unnecessary resource consumption and ensuring that tests execute quickly.



1. **Reflection**

* Testing Techniques Employed:

Unit Testing: Focused on isolating individual components, ensuring they function correctly.

* Testing Techniques Not Used:

Integration Testing: Assessing interactions between different services.

Stress Testing: Evaluating system stability under extreme conditions.

Performance Testing: Measuring system responsiveness and efficiency.

* Characteristics and Implications:

Unit Testing: Ideal for early-stage defect detection in isolated components.

Integration Testing: Crucial for ensuring seamless interactions in complex projects.

Stress and Performance Testing: Vital for applications expecting high usage to guarantee system reliability.

1. **Mindset**

* Caution in Testing:

Example: In the appointment service, cautious testing of date validation was essential to prevent issues related to time-sensitive functionalities.

* Limiting Bias in Code Review:

Example: During task service testing, efforts were made to objectively evaluate logic without being influenced by personal opinions on task priorities.

* Bias Concerns as a Developer:

Example: Testing one's own code may introduce bias, potentially affecting the assessment of implementation choices. In such cases, involving other team members in testing is crucial.

* Importance of Discipline in Quality Commitment

Avoiding Cutting Corners:

Example: Thorough testing before code implementation, as demonstrated in the contact service tests, helps identify and rectify issues early on, preventing future challenges.

Avoiding Technical Debt:

Example: Committing to code quality and testing prevents technical debt. Skipping thorough testing can result in unreliable software and increased maintenance efforts.

1. **Conclusion:**

In conclusion, the unit testing approach for Project One was meticulous, aligning closely with software requirements. The reflection highlights the importance of different testing techniques, the mindset of caution and unbiased evaluation, and the critical role of discipline in maintaining code quality and avoiding technical debt.