Assignment: Task Service with Comprehensive Testing, Static Code Analysis, and Reviews

Objective:
Students will continue refining their Task Service, focusing on testing, static code analysis, and peer reviews. In addition, they will apply concepts from Code Review and Software Review to evaluate code quality, ensuring both automated and manual review processes are well-understood.

Instructions:

1. Refine the Task Service:

- Task Service: Build on your existing Task Service that manages task CRUD operations. Ensure it supports:
    - Create, Read, Update, Delete tasks.
    - Enforce simple business rules (e.g., task title must not be empty, task description should have a minimum length).

Focus on maintaining clean, readable, and efficient code as this will be reviewed by peers.

2. Unit Testing with Mocking:

- Continue using the mocking framework (Mockito, EasyMock, JMockit, or Moq for C#) to mock external dependencies (e.g., database, third-party services).
- Write unit tests that thoroughly cover the core business logic of the Task Service.
- Ensure proper use of mocks to isolate business logic from external systems.
- Achieve at least 75% code coverage with unit tests.

3. Test Design Using Equivalence Partitioning and Boundary Value Analysis:

- Design at least three test cases using Equivalence Partitioning and Boundary Value Analysis:
    - Boundary Value Analysis: Test minimum and maximum lengths for task titles.
    - Equivalence Partitioning: Test valid and invalid task states (e.g., active, completed, invalid).

- Implement these test cases in your unit testing suite and ensure they are covered in your reports.

4. Static Code Analysis with PMD:

- Integrate PMD (or FxCop for C#) into your build process.
- Run PMD to analyze your codebase for issues such as:
    - Code style violations.
    - Potential bugs (e.g., null pointer dereferences).
    - Inefficiencies in code (e.g., unused variables).
- Fix at least three critical issues identified by PMD.
- Submit both the PMD report and the corrected code.

5. Code Coverage with JaCoCo:

- Set up JaCoCo (or Coverlet for C#) to measure code coverage.
- Ensure minimum 75% code coverage and generate a coverage report.
- Review the report to confirm that the core logic and edge cases are adequately tested.
- Submit the JaCoCo report as part of your deliverables.

6. Comprehensive Code Quality with SonarQube (Optional):

- If time allows, integrate SonarQube to get a more in-depth analysis of code quality, detecting code smells, bugs, and duplications.
- Use SonarQube's analysis to fix at least three major issues.
- Submit the SonarQube report along with other deliverables (optional).

7. Peer Code Review (Based on Code Review Article):

- Conduct a code review with another student in your group based on the principles from the Code Review Wikipedia article. Focus on:
    - Code readability and maintainability.
    - Adherence to coding standards.
    - Correct use of mocks in testing.
    - Identification of potential bugs.

- Provide constructive feedback on your partner's code. Ensure feedback is clear, specific, and actionable.
- Use tools like GitHub Pull Requests or GitLab Merge Requests for conducting the review (optional).
- Submit your written feedback and any changes suggested or made based on the review.

8. Software Review (Based on Software Review Article):

- Perform a Software Review on your Task Service codebase. This should cover:
    - Functional correctness: Does the code do what it's supposed to do?
    - Non-functional aspects: Assess performance, maintainability, and testability.
    - Adherence to best practices: Evaluate code structure, documentation, and commenting.
- Reflect on your findings in a 300-word report, noting any improvements made after this review.

9. Reflection on Testing and Code Quality:

- Write a brief reflection (300-400 words) on:
    - The impact of using static code analysis tools like PMD and JaCoCo.
    - The importance of mocking in unit testing for isolating external dependencies.
    - The value of performing both code reviews and software reviews in ensuring quality.
    - How Equivalence Partitioning and Boundary Value Analysis influenced your test design.
- Discuss how these processes helped you identify and fix potential issues early.

Deliverables:

- Refined Task Service source code.
- Unit tests (with mocks) for core functionality.
- PMD report (or equivalent for C#).
- JaCoCo coverage report (minimum 75%).

- Peer code review feedback and any code changes.
- Software review reflection.
- Optional: SonarQube report (if applicable).

Project Summary:

- Task Service API: A CRUD API to manage tasks, with a focus on testability and code quality.
- Focus Areas:
    - Mocking to isolate business logic during testing.
    - Static code analysis for identifying code quality issues early.
    - Code review to assess code clarity and potential bugs.
    - Software review to evaluate overall correctness, maintainability, and performance.
    - Equivalence Partitioning and Boundary Value Analysis to ensure comprehensive testing.

Key Tools and Technologies:

- Java/C# (students can use any language but must find equivalent tools).
- Mockito, EasyMock, JMockit (Java), or Moq (C#) for mocking.
- PMD (Java) or FxCop (C#) for static code analysis.
- JaCoCo (Java) or Coverlet (C#) for code coverage.
- SonarQube for deeper code quality analysis (optional).