Software Architecture Fundamentals & Quality Attributes – JUnit

1. Chosen Software System: JUnit

JUnit is a widely used unit testing framework for Java that enables developers to write and execute automated tests. It follows a test-driven development (TDD) approach, allowing developers to verify the correctness of their code efficiently.

2. Architectural Style Used

JUnit follows a Layered Architecture with a modular component-based design.

- **Test Execution Engine**: Responsible for running test cases and reporting results.
- **Annotations and Reflection Layer**: Uses Java's annotation processing and reflection API to detect and execute test methods.
- **Assertions Library**: Provides a collection of assertion methods to validate expected outcomes.
- **Extension Model (JUnit 5 Jupiter API)**: Supports plug-in extensions for parameterized tests, custom rules, and lifecycle hooks.
- **Reporting Layer**: Generates test execution reports for integration with CI/CD pipelines.

3. How the architecture supports key non-functional quality attributes

Maintainability:

Modular design: Each component is independent, so updates are easier and do not affect the entire system.

Backward compatibility: JUnit 5 maintains compatibility with older JUnit 4 tests through a retro engine, ensuring migration issues are minimized.

Extensibility: The extended model allows customization of test runners, parameterized tests, and lifecycle hooks for increased adaptability.

Performance:

Selective Test Execution: Developers can execute only modified or relevant tests, increasing the efficiency of large projects.

Parallel Execution Support: JUnit 5 allows tests to run in parallel, optimizing performance for large-scale applications.

Lightweight framework: Minimal runtime overhead ensures fast test execution, which is critical for CI/CD workflows.

4. Trade-offs in JUnit's Design

- **Higher Complexity in JUnit 5**: The modular architecture improves flexibility but introduces a learning curve compared to JUnit 4's simpler design.
- **Increased Memory Usage in Large-Scale Testing**: Running thousands of tests in parallel may require significant memory optimization.
- **Reflection Overhead**: Uses Java reflection to discover and execute test cases, which may have a slight performance impact compared to direct method calls.