LO5 CI

1. Functional Requirement Analysis

Functional requirement is "The drone delivery system should accurately deliver orders to the specified location within the set timeframe."

1.1 Requirement Analysis

Requirement Description: The system needs the capability to receive orders, plan flight paths, monitor the drone's status in real-time, and ensure orders are delivered within the scheduled timeframe.

1.2 Intended Code Implementation:

Receiving Orders: The system receives order information through an API interface, including delivery address, order details, and scheduled delivery time.
Planning Flight Path: The system uses a mapping service API to calculate the optimal flight path from the warehouse to the delivery address.
Real-time Monitoring of Drone Status: The system tracks the drone's position in real-time through its GPS signal to ensure flight safety and adherence to the plan.

1.3 Planned Testing:

Test if the drone can receive orders and correctly parse order information.
Test if the system can calculate a valid flight path, considering no-fly zones and weather restrictions.

1.4 Approaches to Improving Through Reviews and Inspections

Code Reviews: Organize code review meetings, inviting team members and other stakeholders to participate, focusing on checking if the code implementation meets the requirement description, especially the logic for processing orders and planning flight paths.

1.5 Automated Static Code Analysis: Utilize static code analysis tools to automatically detect potential coding errors, performance issues, or security vulnerabilities, particularly in modules handling order information and flight path computation.

1.6 Test Case Reviews: Conduct reviews of test cases to ensure coverage of all key functionality points, including order reception, flight path planning, and drone status monitoring.

2. Measurable Attribute Analysis: System Performance

2.1 Attribute Definition

Performance: The system's efficiency in calculating flight paths and processing orders. This includes the speed of response to order requests, the computation time for determining optimal flight paths, and the system's ability to handle a high volume of orders simultaneously.

2.2 Intended Code Implementation

Order Processing: Implement asynchronous processing of orders to allow the system to receive and queue multiple orders without significant delays.

Flight Path Calculation: Utilize optimized algorithms for calculating flight paths, such as Dijkstra's or A* search algorithms, to ensure quick and efficient pathfinding.

Load Handling: Design the system with scalability in mind, using cloud-based services or load balancers to distribute processing load during peak times.

2.3 Planned Testing for Performance

Load Testing: Simulate high-volume order scenarios to test the system's capacity to handle multiple orders simultaneously without performance degradation.

Stress Testing: Incrementally increase the load beyond the expected maximum to identify the system's breaking point and observe how it recovers from failure.

Performance Benchmarking: Establish baseline performance metrics for order processing time and flight path calculation time under normal conditions. Compare these metrics under test conditions to evaluate performance impact.

2.4 Approaches to Improving Through Reviews and Inspections

Code Optimization Reviews: Conduct focused reviews on the code sections responsible for order processing and flight path calculations to identify potential optimizations, such as algorithm improvements or more efficient data structures.

Architecture Reviews: Evaluate the system architecture for its scalability and efficiency in handling peak loads. This can include reviewing the use of cloud services, load balancing strategies, and database performance.

Automated Performance Testing: Implement automated tests that can be run regularly as part of the CI/CD pipeline to continuously monitor performance against established benchmarks. Utilize tools that can simulate user behavior and measure response times and system throughput.

3. Defect Comparison

3.1 Defect Comparison

Deficiencies from Functional Requirement Analysis

Lack of real-world testing environments, leading to an inability to fully simulate the system's performance under actual operational conditions.

Insufficient physical device testing, especially lacking support for testing across multiple platforms (such as Linux systems and Android devices).

Stress testing utilized synthetic data, which may not accurately reflect real user behavior and data patterns.

Deficiencies from Measurable Attribute Analysis
Performance testing might not cover all high-load scenarios, especially the system's performance under extreme conditions.
Performance benchmarks might be based on idealized or standardized conditions, insufficient to reflect the complexity and unpredictability of the real world.
Automated testing coverage may be inadequate, particularly for performance and load testing.

3.2 Similarities and Differences Between Deficiencies

Similarities:
Both reveal limitations regarding testing environments and data, which could affect the accuracy and reliability of test results.
A common deficiency in both functional requirement and measurable attribute analysis is the insufficiency of automated testing, indicating a need to further expand the scope and depth of test automation.

Differences:
Deficiencies in functional requirement analysis are more concerned with the realism and diversity of testing environments, whereas deficiencies in measurable attribute analysis focus on the comprehensiveness of performance testing and the setting of benchmarks.
Deficiencies in measurable attribute analysis involve the specific quantification of performance indicators, requiring more detailed methods of performance evaluation and benchmarking, while deficiencies in functional requirement analysis are more about the design of test cases and environment configuration.

Approaches to Improvement Through Reviews and Inspections
For the limitations regarding testing environments and data, reviews and improvements can include introducing more real-world testing scenarios and using real user data.
Increase testing support for physical devices and multiple platform environments by conducting specialized hardware tests and cross-platform compatibility checks to address deficiencies.
Expand the coverage of automated testing, especially in performance and load testing, utilizing automation tools to regularly execute and monitor performance metrics.